Microsoft Malware detection

1.Business/Real-world Problem

1.1. What is Malware?

The term malware is a contraction of malicious software. Put simply, malware is any piece of software that was written with the intent of doing harm to data, devices or to people.
Source: https://www.avg.com/en/signal/what-is-malware

1.2. Problem Statement

In the past few years, the malware industry has grown very rapidly that, the syndicates invest heavily in technologies to evade traditional protection, forcing the anti-malware groups/communities to build more robust softwares to detect and terminate these attacks. The major part of protecting a computer system from a malware attack is to **identify whether a given piece of file/software is a malware.**

1.3 Source/Useful Links

Microsoft has been very active in building anti-malware products over the years and it runs it's anti-malware utilities over 150 million computers around the world. This generates tens of millions of daily data points to be analyzed as potential malware. In order to be effective in analyzing and classifying such large amounts of data, we need to be able to group them into groups and identify their respective families.

This dataset provided by Microsoft contains about 9 classes of malware. ,

Source: https://www.kaggle.com/c/malware-classification

1.4. Real-world/Business objectives and constraints.

1. **Minimize multi-class error.**
2. **Multi-class probability estimates.**
3. **Malware detection should not take hours and block the user's computer. It should fininsh in a few seconds or a minute.**

2. Machine Learning Problem

2.1. Data

2.1.1. Data Overview

- **Source : https://www.kaggle.com/c/malware-classification/data**
- **For every malware, we have two files**
  1. **.asm file (read more: https://www.reviversoft.com/file-extensions/asm)**
  2. **.bytes file (the raw data contains the hexadecimal representation of the file's binary content, without the PE header)**
- **Total train dataset consist of 200GB data out of which 50Gb of data is .bytes files and 150GB of data is .asm files:**
- **Lots of Data for a single-box/computer.**
- **There are total 10,868 .bytes files and 10,868 asm files total 21,736 files**
- **There are 9 types of malwares (9 classes) in our give data**
- **Types of Malware:**
  1. **Ramnit**

1. Kelihos_ver3
2. Lollipop
3. Kelihos_ver3
4. Vundo
5. Simda
6. Tracur
7. Kelihos_ver1
8. Obfuscator.ACY
9. Gatak

2.1.2. Example Data Point

.asm file

```
.text:00401000                                  assume es:nothing, ss:nothing, ds:_data,
s:nothing, gs:nothing
.text:00401000 56                               push    esi
.text:00401001 8D 44 24 08                          lea     eax, [esp+8]
.text:00401005 50                               push    eax
.text:00401006 8B F1                                mov     esi, ecx
.text:00401008 E8 1C 1B 00 00                       call    ??
0exception@std@@QAE@ABQBD@Z ; std::exception::exception(char const * const &)
.text:0040100D C7 06 08 BB 42 00                    mov     dword ptr [esi], offset o
f_42BB08
.text:00401013 8B C6                                mov     eax, esi
.text:00401015 5E                               pop     esi
.text:00401016 C2 04 00                             retn    4
.text:00401016                              ; ---------------------------------------------
-------------------------
.text:00401019 CC CC CC CC CC CC CC                 align 10h
.text:00401020 C7 01 08 BB 42 00                    mov     dword ptr [ecx], offset o
f_42BB08
.text:00401026 E9 26 1C 00 00                       jmp     sub_402C51
.text:00401026                              ; ---------------------------------------------
-------------------------
.text:0040102B CC CC CC CC CC                       align 10h
.text:00401030 56                               push    esi
.text:00401031 8B F1                                mov     esi, ecx
.text:00401033 C7 06 08 BB 42 00                    mov     dword ptr [esi], offset o
f_42BB08
.text:00401039 E8 13 1C 00 00                       call    sub_402C51
.text:0040103E F6 44 24 08 01                       test    byte ptr [esp+8], 1
.text:00401043 74 09                                jz      short loc_40104E
.text:00401045 56                               push    esi
.text:00401046 E8 6C 1E 00 00                       call    ??3@YAXPAX@Z    ; operato
delete(void *)
.text:0040104B 83 C4 04                             add     esp, 4
.text:0040104E
.text:0040104E                              loc_40104E:                 ; CODE XREF:
.text:00401043 j
.text:0040104E 8B C6                                mov     eax, esi
.text:00401050 5E                               pop     esi
.text:00401051 C2 04 00                             retn    4
.text:00401051                              ; ---------------------------------------------
-------------------------
```

.bytes file

```
00401000 00 00 80 40 40 28 00 1C 02 42 00 C4 00 20 04 20
00401010 00 00 20 09 2A 02 00 00 00 00 8E 10 41 0A 21 01
00401020 40 00 02 01 00 90 21 00 32 40 00 1C 01 40 C8 18
00401030 40 82 02 63 20 00 00 09 10 01 02 21 00 82 00 04
00401040 82 20 08 83 00 08 00 00 00 00 02 00 60 80 10 80
00401050 18 00 00 20 A9 00 00 00 00 04 04 78 01 02 70 90
```

```
00401060 00 02 00 08 20 12 00 00 00 40 10 00 80 00 40 19
00401070 00 00 00 00 11 20 80 04 80 10 00 20 00 00 25 00
00401080 00 00 01 00 00 04 00 10 02 C1 80 80 00 20 20 00
00401090 08 A0 01 01 44 28 00 00 08 10 20 00 02 08 00 00
004010A0 00 40 00 00 00 34 40 40 00 04 00 08 80 08 00 08
004010B0 10 00 40 00 68 02 40 04 E1 00 28 14 00 08 20 0A
004010C0 06 01 02 00 40 00 00 00 00 00 00 20 00 02 00 04
004010D0 80 18 90 00 00 10 A0 00 45 09 00 10 04 40 44 82
004010E0 90 00 26 10 00 00 04 00 82 00 00 00 20 40 00 00
004010F0 B4 00 00 40 00 02 20 25 08 00 00 00 00 00 00 00
00401100 08 00 00 50 00 08 40 50 00 02 06 22 08 85 30 00
00401110 00 80 00 80 60 00 09 00 04 20 00 00 00 00 00 00
00401120 00 82 40 02 00 11 46 01 4A 01 8C 01 E6 00 86 10
00401130 4C 01 22 00 64 00 AE 01 EA 01 2A 11 E8 10 26 11
00401140 4E 11 8E 11 C2 00 6C 00 0C 11 60 01 CA 00 62 10
00401150 6C 01 A0 11 CE 10 2C 11 4E 10 8C 00 CE 01 AE 01
00401160 6C 10 6C 11 A2 01 AE 00 46 11 EE 10 22 00 A8 00
00401170 EC 01 08 11 A2 01 AE 10 6C 00 6E 00 AC 11 8C 00
00401180 EC 01 2A 10 2A 01 AE 00 40 00 C8 10 48 01 4E 11
00401190 0E 00 EC 11 24 10 4A 10 04 01 C8 11 E6 01 C2 00
```

2.2. Mapping the real-world problem to an ML problem

2.2.1. Type of Machine Learning Problem

There are nine different classes of malware that we need to classify a given a data point => Multi class classification problem

2.2.2. Performance Metric

Source: https://www.kaggle.com/c/malware-classification#evaluation

Metric(s):

- **Multi class log-loss**
- **Confusion matrix**

2.2.3. Machine Learing Objectives and Constraints

Objective: Predict the probability of each data-point belonging to each of the nine classes.

Constraints:

- **Class probabilities are needed.**
- **Penalize the errors in class probabilites => Metric is Log-loss.**
- **Some Latency constraints.**

2.3. Train and Test Dataset

Split the dataset randomly into three parts train, cross validation and test with 64%,16%, 20% of data respectively

2.4. Useful blogs, videos and reference papers

http://blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/
https://arxiv.org/pdf/1511.04317.pdf
First place solution in Kaggle competition: https://www.youtube.com/watch?v=VLQTRILGz5Y

https://github.com/dchad/malware-detection
http://vizsec.org/files/2011/Nataraj.pdf
https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EeInEjvvuQg2nu_pIB6ua?dl=0
" Cross validation is more trustworthy than domain knowledge."

3. Exploratory Data Analysis

In [6]:

```
import warnings
warnings.filterwarnings("ignore")
import shutil
import os
import pandas as pd
import matplotlib
matplotlib.use(u'nbAgg')
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pickle
from sklearn.manifold import TSNE
from sklearn import preprocessing
import pandas as pd
from multiprocessing import Process# this is used for multithreading
import multiprocessing
import codecs# this is used for file operations
import random as r
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

In [5]:

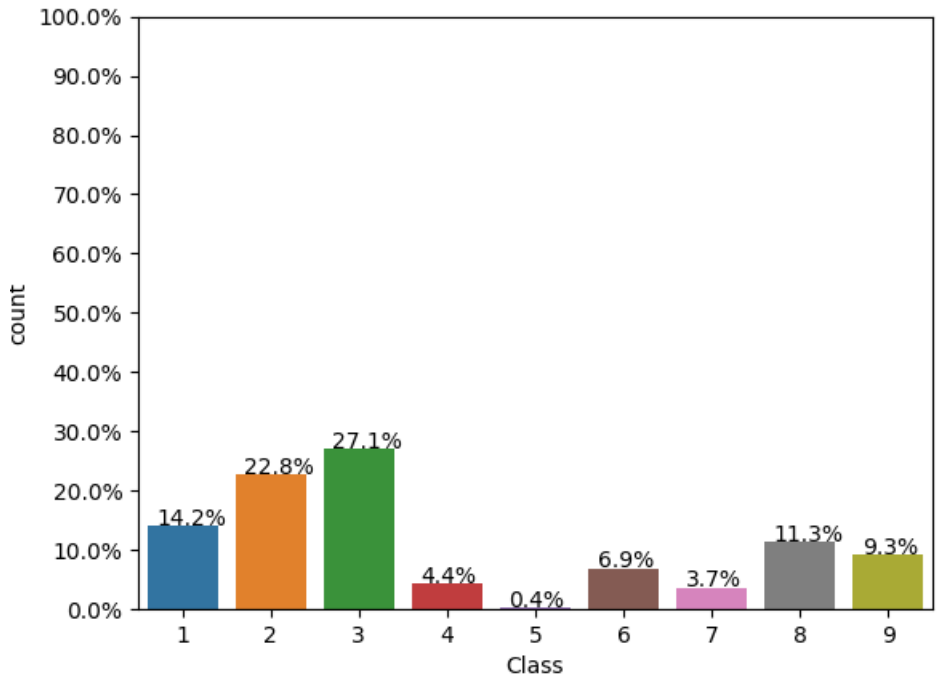
```
source = 'train'
destination = 'byteFiles'

# we will check if the folder 'byteFiles' exists if it not there we will create a folder with the
same name
if not os.path.isdir(destination):
    os.makedirs(destination)

# if we have folder called 'train' (train folder contains both .asm files and .bytes files) we wil
l rename it 'asmFiles'
# for every file that we have in our 'asmFiles' directory we check if it is ending with .bytes, if
yes we will move it to
# 'byteFiles' folder

# so by the end of this snippet we will separate all the .byte files and .asm files
if os.path.isdir(source):
    os.rename(source,'asmFiles')
    source='asmFiles'
    data_files = os.listdir(source)
    for file in data_files:
        if (file.endswith("bytes")):
            shutil.move(source+file,destination)
```

3.1. Distribution of malware classes in whole data set

In [7]:

```
Y=pd.read_csv("./Microsoft Malware Detection/trainLabels.csv")
total = len(Y)*1.
ax=sns.countplot(x="Class", data=Y)
for p in ax.patches:
```

```
        ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.1, p.get_height()+5))

#put 11 ticks (therefore 10 steps), from 0 to the total number of rows in the dataframe
ax.yaxis.set_ticks(np.linspace(0, total, 11))

#adjust the ticklabel to the desired format, without changing the position of the ticks.
ax.set_yticklabels(map('{:.1f}%'.format, 100*ax.yaxis.get_majorticklocs()/total))
plt.show()
```



3.2. Feature extraction

3.2.1 File size of byte files as a feature

In [8]:

```
files=os.listdir('./Microsoft Malware Detection/byteFiles')
filenames=Y['Id'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_nlink=1,
st_uid=0, st_gid=0,
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519638522)
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.htm
    statinfo=os.stat('./Microsoft Malware Detection/byteFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
data_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (data_size_byte.head())
```

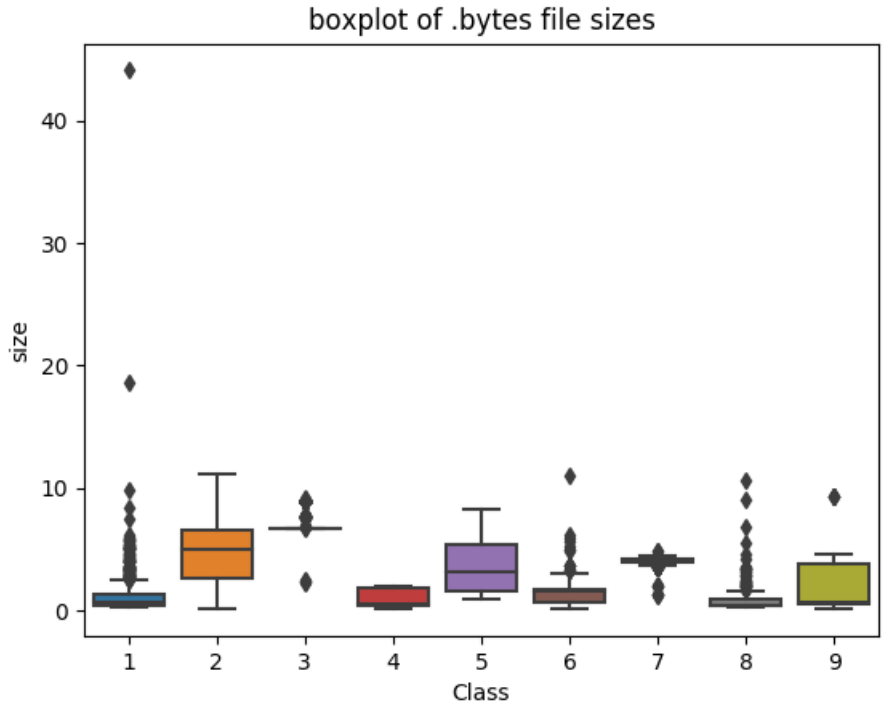
```
   Class                       ID      size
0      2  C4iJQzFIRWB19USfsaVw  5.636719
1      3  1V2h8lnzeGiuxmHR9k5Q  6.703125
```

```
2       4  7KHscjvztoka0QpqYFxb  0.199219
3       9  2qpZmcvFs4LCBNi9IX6H  0.773438
4       4  81jMeyEZiOpJkCuBlDYg  1.839844
```

3.2.2 box plots of file size (.byte files) feature

In [9]:

```
#boxplot of byte files
ax = sns.boxplot(x="Class", y="size", data=data_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```



3.2.3 feature extraction from byte files

In [11]:

```
#removal of addres from byte files
# contents of .byte files
# ----------------
#00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
#--------------------
#we remove the starting address 00401000

files = os.listdir('byteFiles')
filenames=[]
array=[]
for file in files:
    if(file.endswith("bytes")):
        file=file.split('.')[0]
        text_file = open('byteFiles/'+file+".txt", 'w+')
        file = file+'.bytes'
        with open('byteFiles/'+file,"r") as fp:
            lines=""
            for line in fp:
                a=line.rstrip().split(" ")[1:]
                b=' '.join(a)
                b=b+"\n"
                text_file.write(b)
            fp.close()
            os.remove('byteFiles/'+file)
        text_file.close()
```

```
files = os.listdir('byteFiles')
filenames2=[]
feature_matrix = np.zeros((len(files),257),dtype=int)
k=0
```

In [ ]:

```

```

In [ ]:

```

```

In [12]:

```
#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
byte_feature_file=open('result.csv','w+')
byte_feature_file.write("ID,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b
,1d,1e,1f,20,21,22,23,24,25,26,27,28,29,2a,2b,2c,2d,2e,2f,30,31,32,33,34,35,36,37,38,39,3a,3b,3c,3d
3f,40,41,42,43,44,45,46,47,48,49,4a,4b,4c,4d,4e,4f,50,51,52,53,54,55,56,57,58,59,5a,5b,5c,5d,5e,5f,
1,62,63,64,65,66,67,68,69,6a,6b,6c,6d,6e,6f,70,71,72,73,74,75,76,77,78,79,7a,7b,7c,7d,7e,7f,80,81,8
,84,85,86,87,88,89,8a,8b,8c,8d,8e,8f,90,91,92,93,94,95,96,97,98,99,9a,9b,9c,9d,9e,9f,a0,a1,a2,a3,a4
a6,a7,a8,a9,aa,ab,ac,ad,ae,af,b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,ba,bb,bc,bd,be,bf,c0,c1,c2,c3,c4,c5,c6,
8,c9,ca,cb,cc,cd,ce,cf,d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,da,db,dc,dd,de,df,e0,e1,e2,e3,e4,e5,e6,e7,e8,e
,eb,ec,ed,ee,ef,f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??")
for file in files:
    filenames2.append(file)
    byte_feature_file.write(file+",")
    if(file.endswith("txt")):
        with open('byteFiles/'+file,"r") as byte_flie:
            for lines in byte_flie:
                line=lines.rstrip().split(" ")
                for hex_code in line:
                    if hex_code=='??':
                        feature_matrix[k][256]+=1
                    else:
                        feature_matrix[k][int(hex_code,16)]+=1
        byte_flie.close()
    for i in feature_matrix[k]:
        byte_feature_file.write(str(i)+",")
    byte_feature_file.write("\n")

    k += 1

byte_feature_file.close()
```

In [10]:

```
byte_features.dropna(axis=1, how='all',inplace=True)
byte_features.rename(columns = {"??C4iJQzFIRWB19USfsaVw.txt":"??"}, inplace=True)
byte_features['ID'] = byte_features['ID'].str.replace('\.txt','')
byte_features
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-a55277a7f49d> in <module>
----> 1 byte_features.dropna(axis=1, how='all',inplace=True)
      2 byte_features.rename(columns = {"??C4iJQzFIRWB19USfsaVw.txt":"??"}, inplace=True)
      3 byte_features['ID'] = byte_features['ID'].str.replace('\.txt','')
      4 byte_features

NameError: name 'byte_features' is not defined
```

In [18]:

```
data_size_byte
```

Out[18]:

	Class	ID	size
0	2	C4iJQzFIRWB19USfsaVw	5.636719
1	3	1V2h8lnzeGiuxmHR9k5Q	6.703125
2	4	7KHscjvztoka0QpqYFxb	0.199219
3	9	2qpZmcvFs4LCBNi9lX6H	0.773438
4	4	81jMeyEZiOpJkCuBlDYg	1.839844
...	...	...	...
10863	2	hZfoeEcsTJRnNLIF5wG0	6.281250
10864	3	dMr6GYzt8pOD5cQTi2VS	6.703125
10865	3	eT5zyaVgSRMwGc2oQfHW	6.703125
10866	3	1YnwQBXNlKPzcdGWxSA6	6.703125
10867	3	9pVfLJrSuU8iANWPFM15	6.703125

10868 rows × 3 columns

mk

In [20]:

```
result = pd.merge(byte_features, data_size_byte,on='ID', how='left')
byte_features.to_csv("./Microsoft Malware Detection/result.csv")
result.head()
```

Out[20]:

	ID	0	1	2	3	4	5	6	7	8	...	f9	fa	fb	fc	fd	fe	ff
0	1V2h8lnzeGiuxmHR9k5Q	8391	4211	3108	3270	3072	3087	3100	3143	3195	...	3181	3109	3095	3068	3068	3049	5103
1	7KHscjvztoka0QpqYFxb	2585	267	220	267	276	217	199	170	236	...	214	188	195	201	188	201	241
2	2qpZmcvFs4LCBNi9lX6H	101076	1362	597	879	1029	973	487	488	671	...	1457	440	417	597	393	575	2713
3	81jMeyEZiOpJkCuBlDYg	3745	1083	1071	1205	1386	1052	1017	1017	1088	...	1014	1061	1043	1099	1031	1052	1330
4	5H40FbQlckCMa3jguwS9	8234	4230	3079	3223	3158	3173	2886	3003	3232	...	3004	3077	3050	3060	3088	3085	5211

5 rows × 260 columns

In [11]:

```
byte_features=pd.read_csv("./Microsoft Malware Detection/result.csv")
print (byte_features.head())
```

```
   Unnamed: 0                   ID       0     1     2     3     4     5  \
0           0  1V2h8lnzeGiuxmHR9k5Q    8391  4211  3108  3270  3072  3087
1           1  7KHscjvztoka0QpqYFxb    2585   267   220   267   276   217
2           2  2qpZmcvFs4LCBNi9IX6H  101076  1362   597   879  1029   973
3           3  81jMeyEZiOpJkCuBlDYg    3745  1083  1071  1205  1386  1052
4           4  5H40FbQlckCMa3jguwS9    8234  4230  3079  3223  3158  3173

      6     7  ...    f7    f8    f9    fa    fb    fc    fd    fe    ff  \
0  3100  3143  ...  3070  3026  3181  3109  3095  3068  3068  3049  5103
1   199   170  ...   243   214   214   188   195   201   188   201   241
2   487   488  ...   445   636  1457   440   417   597   393   575  2713
3  1017  1017  ...  1085  1053  1014  1061  1043  1099  1031  1052  1330
4  2886  3003  ...  3037  3058  3004  3077  3050  3060  3088  3085  5211

        ??
0  1535644
1    12856
2     2116
3   365728
4  1536156

[5 rows x 259 columns]
```

In [12]:

```
result=byte_features
```

In [13]:

```
# https://stackoverflow.com/a/29651514
def normalize(df):
    result1 = df.copy()
    for feature_name in df.columns:
        if (str(feature_name) != str('ID') and str(feature_name)!=str('Class')):
            max_value = df[feature_name].max()
            min_value = df[feature_name].min()
            result1[feature_name] = (df[feature_name] - min_value) / (max_value - min_value)
    return result1
result = normalize(result)
```

In [ ]:

```

```

In [7]:

```
from sklearn.externals import joblib
```

In [8]:

```
joblib.dump(result, 'result.pkl')
```

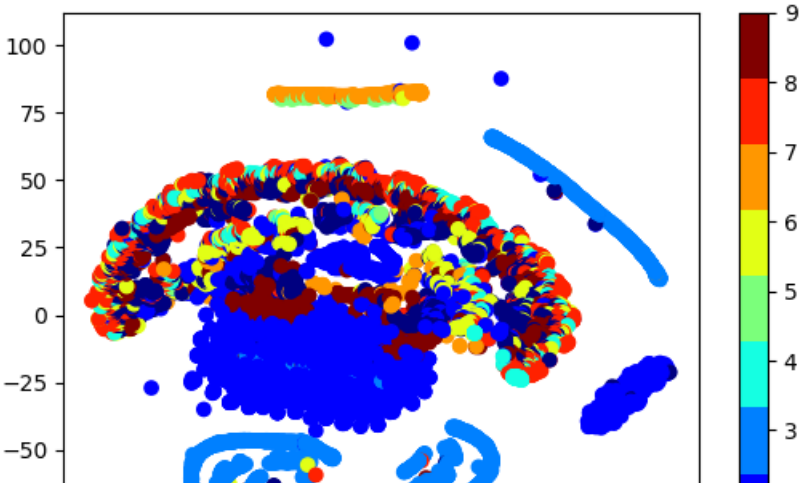
Out[8]:

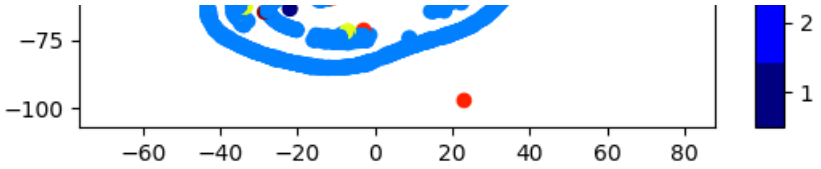
```
['result.pkl']
```

3.2.4 Multivariate Analysis

In [11]:

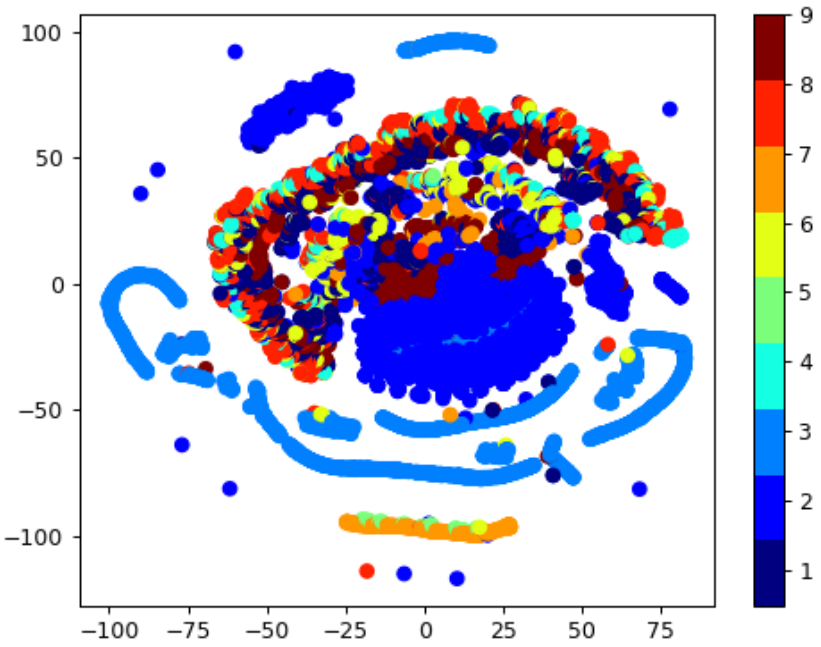
```
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```





In [15]:

```
#this is with perplexity 30
xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



Train Test split

In [9]:

```
data_y = result['Class']
# split the data into test and train by maintaining same distribution of output varaible 'y_true'
[stratify=y_true]
X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID','Class'], axis=1), data_y,str
atify=data_y,test_size=0.20)
# split the train data into train and cross validation by maintaining same distribution of output
varaible 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_size=0.20)
```

In [10]:

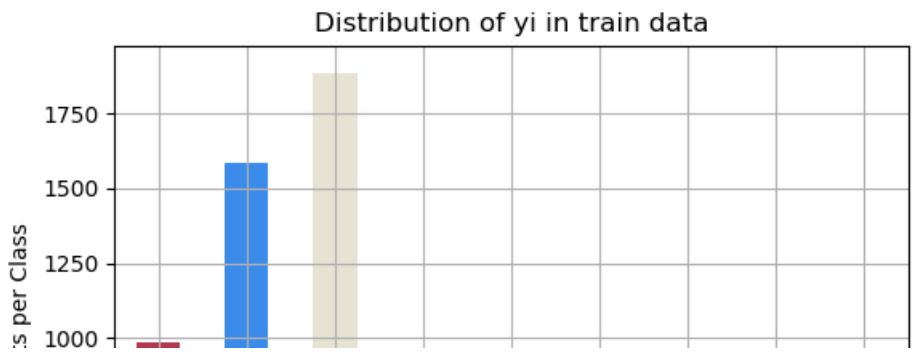
```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

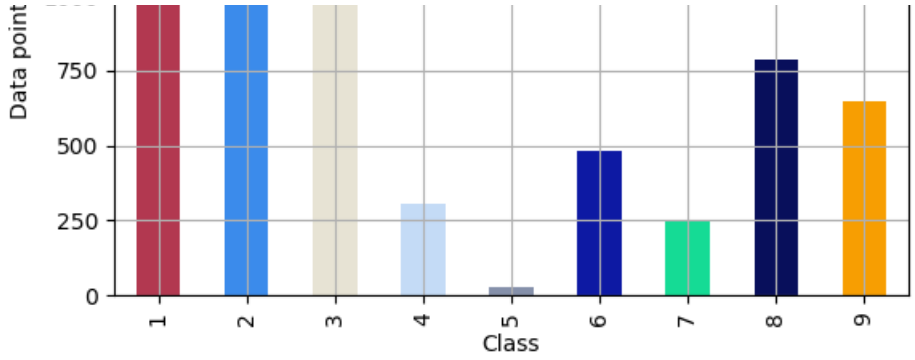
```
Number of data points in train data: 6955
Number of data points in test data: 2174
```

```
Number of data points in cross validation data: 1739
```

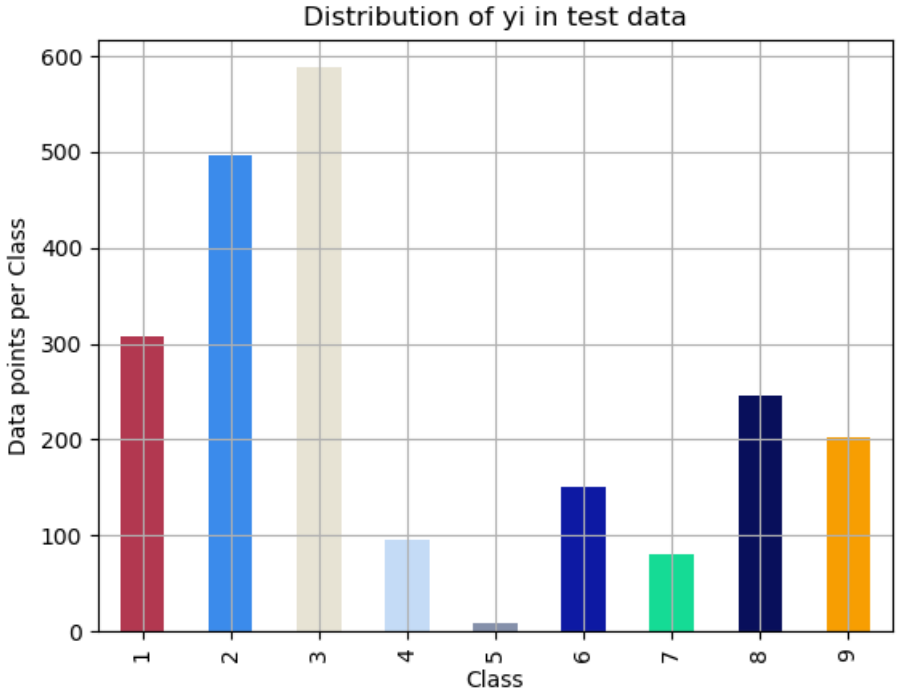
In [11]:

```
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = y_train.value_counts().sortlevel()
test_class_distribution = y_test.value_counts().sortlevel()
cv_class_distribution = y_cv.value_counts().sortlevel()

my_colors = ['#b23850', '#3b8beb', '#e7e3d4', '#c4dbf6', '#8590aa', '#0d19a3', '#15db95',
'#080f5b', '#f79e02']
train_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i], '(', np.ro
und((train_class_distribution.values[i]/y_train.shape[0]*100), 3), '%)')


print('-'*80)
my_colors = ['#b23850', '#3b8beb', '#e7e3d4', '#c4dbf6', '#8590aa', '#0d19a3', '#15db95',
'#080f5b', '#f79e02']
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i], '(', np.rou
nd((test_class_distribution.values[i]/y_test.shape[0]*100), 3), '%)')

print('-'*80)
my_colors = ['#b23850', '#3b8beb', '#e7e3d4', '#c4dbf6', '#8590aa', '#0d19a3', '#15db95',
'#080f5b', '#f79e02']
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i], '(', np.round
((cv_class_distribution.values[i]/y_cv.shape[0]*100), 3), '%)')
```

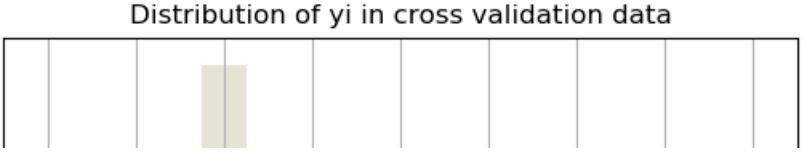


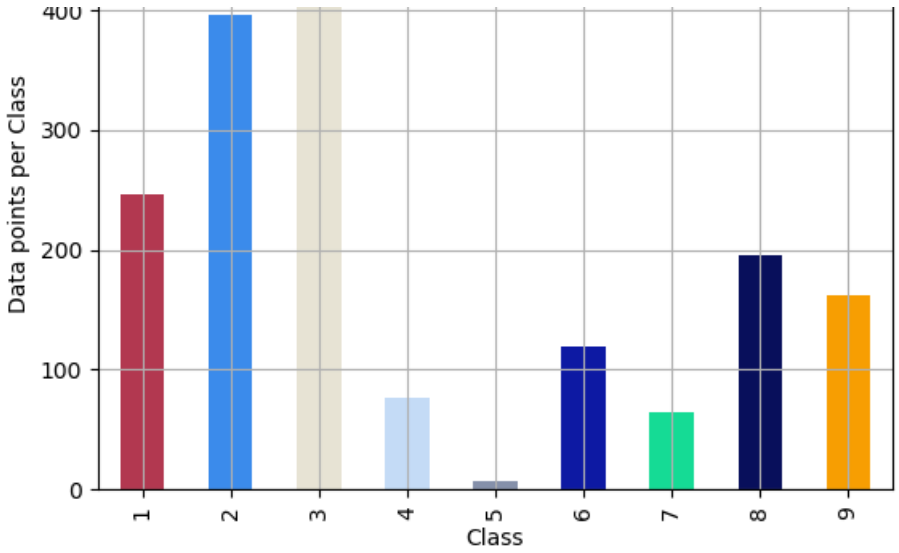



```
Number of data points in class 3 : 1883 ( 27.074 %)
Number of data points in class 2 : 1586 ( 22.804 %)
Number of data points in class 1 : 986 ( 14.177 %)
Number of data points in class 8 : 786 ( 11.301 %)
Number of data points in class 9 : 648 ( 9.317 %)
Number of data points in class 6 : 481 ( 6.916 %)
Number of data points in class 4 : 304 ( 4.371 %)
Number of data points in class 7 : 254 ( 3.652 %)
Number of data points in class 5 : 27 ( 0.388 %)
--------------------------------------------------------------------------------
```




```
Number of data points in class 3 : 588 ( 27.047 %)
Number of data points in class 2 : 496 ( 22.815 %)
Number of data points in class 1 : 308 ( 14.167 %)
Number of data points in class 8 : 246 ( 11.316 %)
Number of data points in class 9 : 203 ( 9.338 %)
Number of data points in class 6 : 150 ( 6.9 %)
Number of data points in class 4 : 95 ( 4.37 %)
Number of data points in class 7 : 80 ( 3.68 %)
Number of data points in class 5 : 8 ( 0.368 %)
--------------------------------------------------------------------------------
```






```
Number of data points in class 3 : 471 ( 27.085 %)
Number of data points in class 2 : 396 ( 22.772 %)
Number of data points in class 1 : 247 ( 14.204 %)
Number of data points in class 8 : 196 ( 11.271 %)
Number of data points in class 9 : 162 ( 9.316 %)
Number of data points in class 6 : 120 ( 6.901 %)
Number of data points in class 4 : 76 ( 4.37 %)
Number of data points in class 7 : 64 ( 3.68 %)
Number of data points in class 5 : 7 ( 0.403 %)
```

In [49]:

```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points ",(len(test_y)-np.trace(C))/len(test_y)*100)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that column

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
diamensional array
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                           [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                           [3/7, 4/7]]
    # sum of row elements = 1

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
diamensional array
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]

    labels = [1,2,3,4,5,6,7,8,9]
    cmap=sns.light_palette("green")
    # representing A in heatmap format
    print("-"*50, "Confusion matrix", "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

```
    print("-"*50, "Precision matrix", "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of columns in precision matrix",B.sum(axis=0))

    # representing B in heatmap format
    print("-"*50, "Recall matrix"    , "-"*50)
    plt.figure(figsize=(10,5))
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
    print("Sum of rows in precision matrix",A.sum(axis=1))
```

4. Machine Learning Models

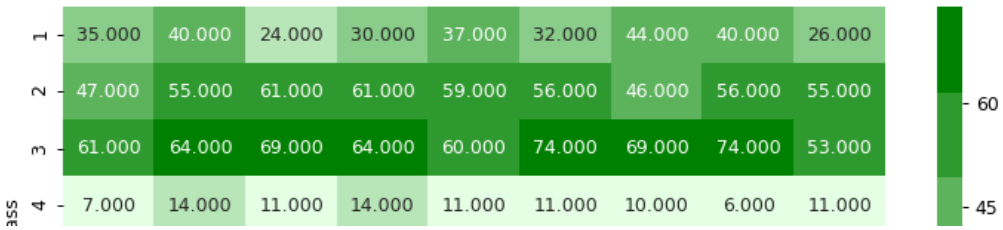
4.1. Machine Leaning Models on bytes files

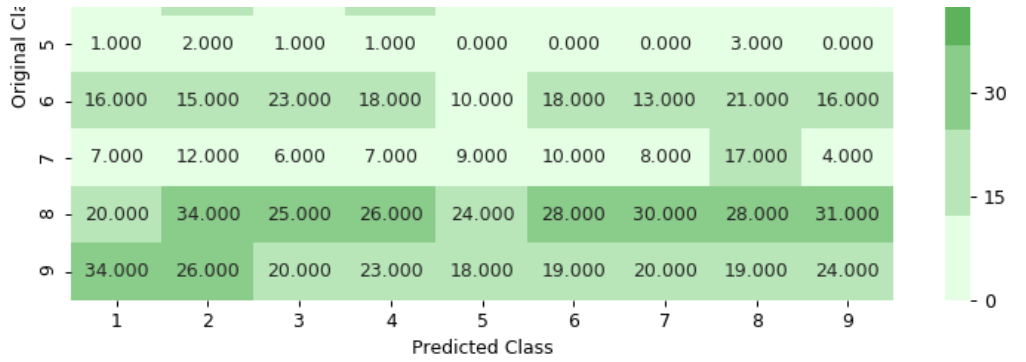
4.1.1. Random Model

In [20]:

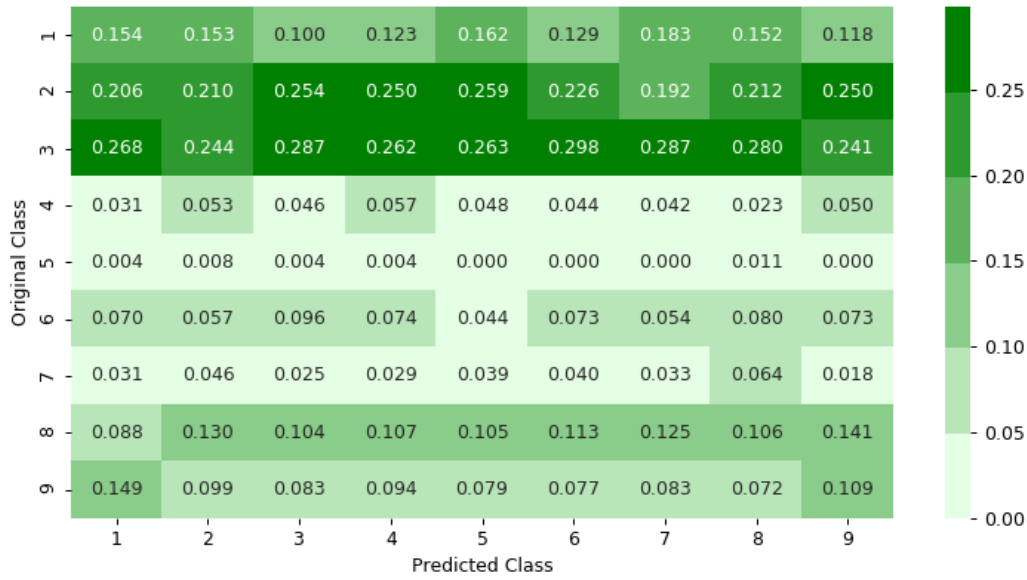
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039

test_data_len = X_test.shape[0]
cv_data_len = X_cv.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicted_y, eps=1e-
15))


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-15))

predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)
```

```
Log loss on Cross Validation Data using Random Model 2.4987116946656167
Log loss on Test Data using Random Model 2.4553327958473936
Number of misclassified points  88.45446182152715
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```



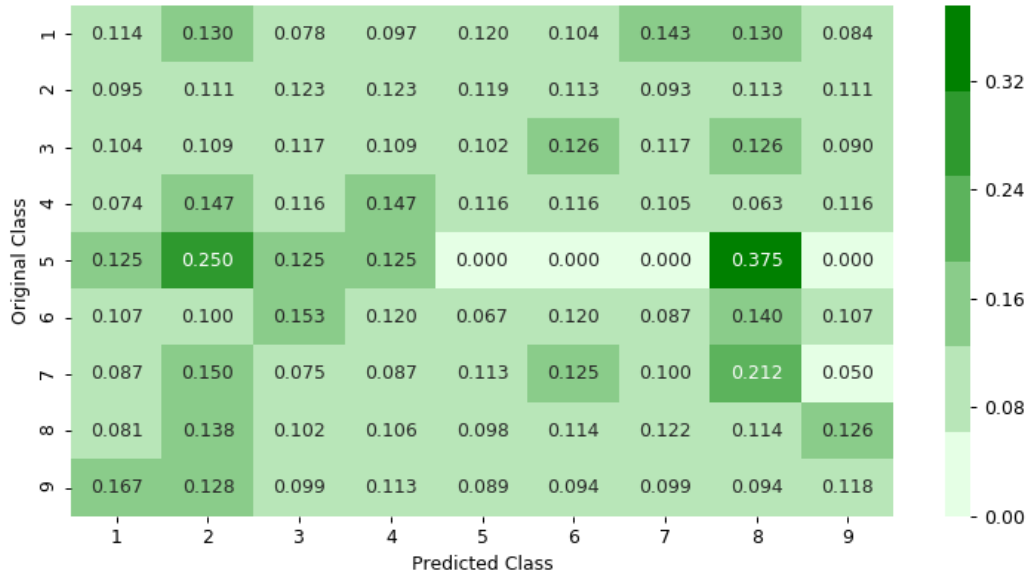


---------- **Precision matrix** ----------



Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]

---------- **Recall matrix** ----------



Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]

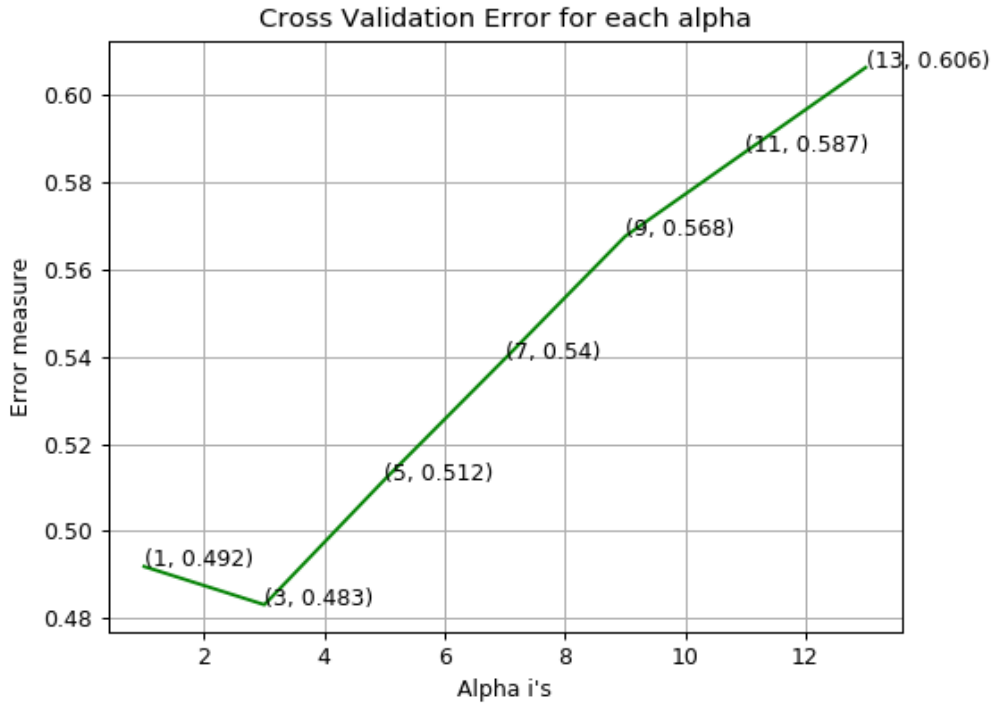
4.1.2. K Nearest Neighbour Classification

In [21]:

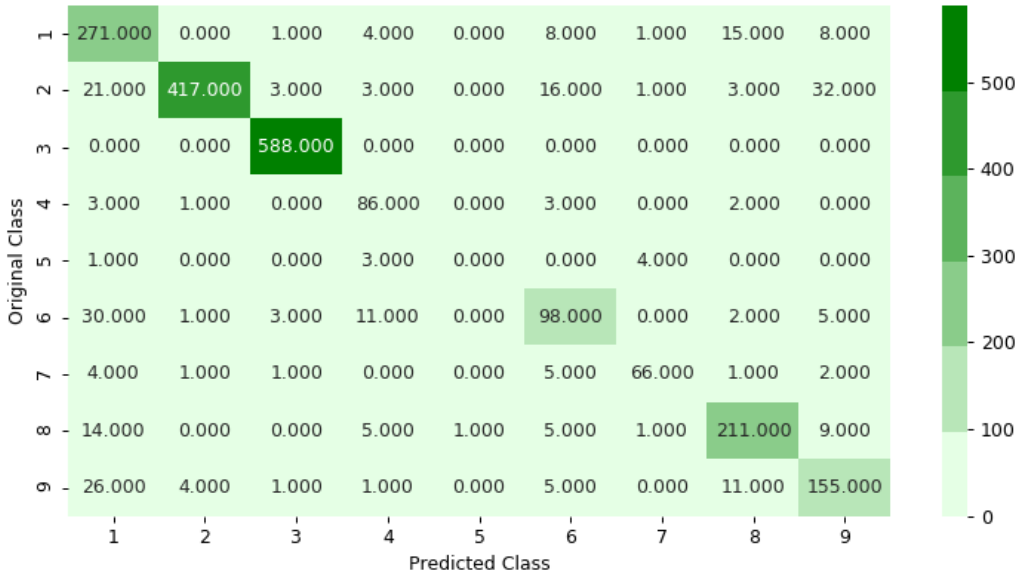
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#--------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-ne
ighbors-geometric-intuition-with-a-toy-example-1/
#--------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
#-----------------------------------
# video link:
#-----------------------------------

alpha = [x for x in range(1, 15, 2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=k_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for k = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

k_cfl=KNeighborsClassifier(n_neighbors=alpha[best_alpha])
k_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train
, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
```

```
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

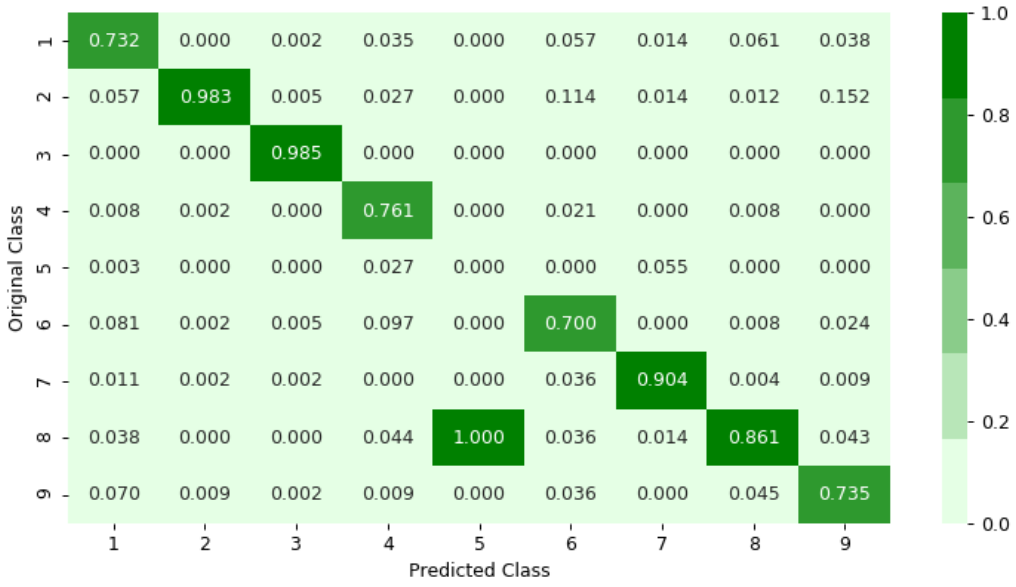
```
log_loss for k =  1 is 0.49188045368463196
log_loss for k =  3 is 0.483116902642161
log_loss for k =  5 is 0.5118350087441232
log_loss for k =  7 is 0.5395490778512431
log_loss for k =  9 is 0.5676371813660702
log_loss for k =  11 is 0.5870170308367498
log_loss for k =  13 is 0.606375118318671
```



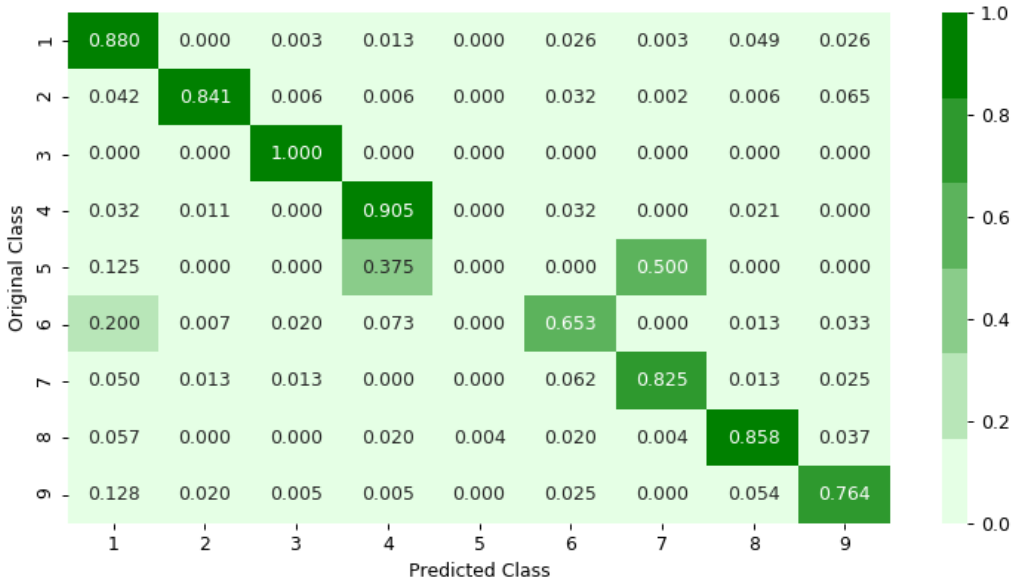

```
For values of best alpha =  3 The train log loss is: 0.29320594139515405
For values of best alpha =  3 The cross validation log loss is: 0.483116902642161
For values of best alpha =  3 The test log loss is: 0.4851954874286108
Number of misclassified points  12.97148114075437
---------------------------------------------------- Confusion matrix -----------------------------
----------------
```



```
------------------------------------------------- Precision matrix -----------------------------
----------------
```




```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
------------------------------------------------- Recall matrix --------------------------------
-------------
```




```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

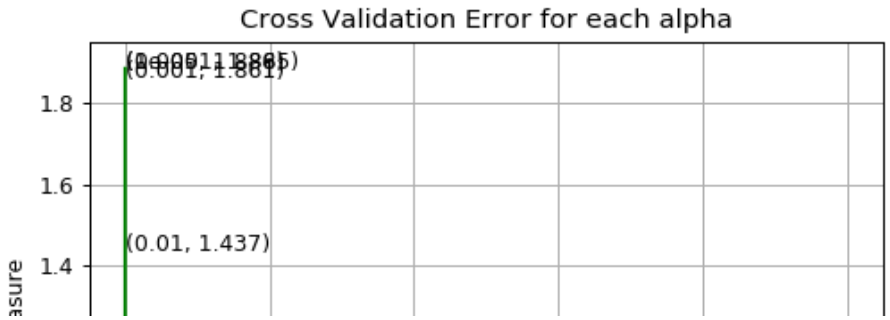
4.1.3. Logistic Regression

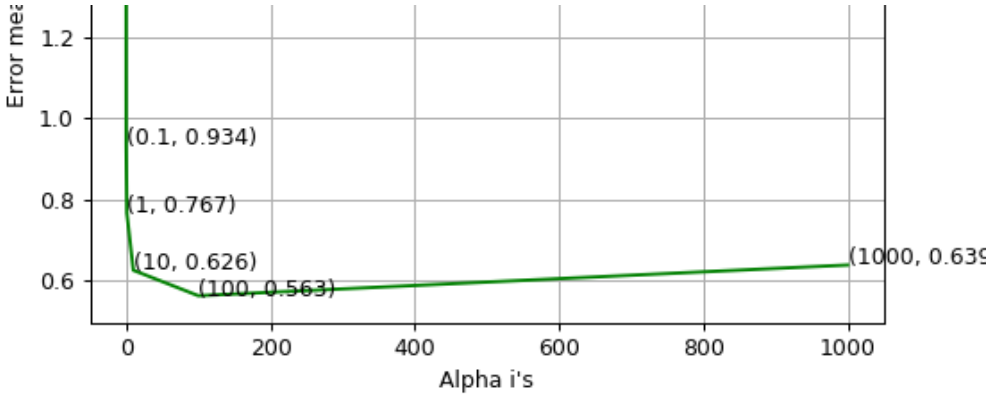
In [22]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
```

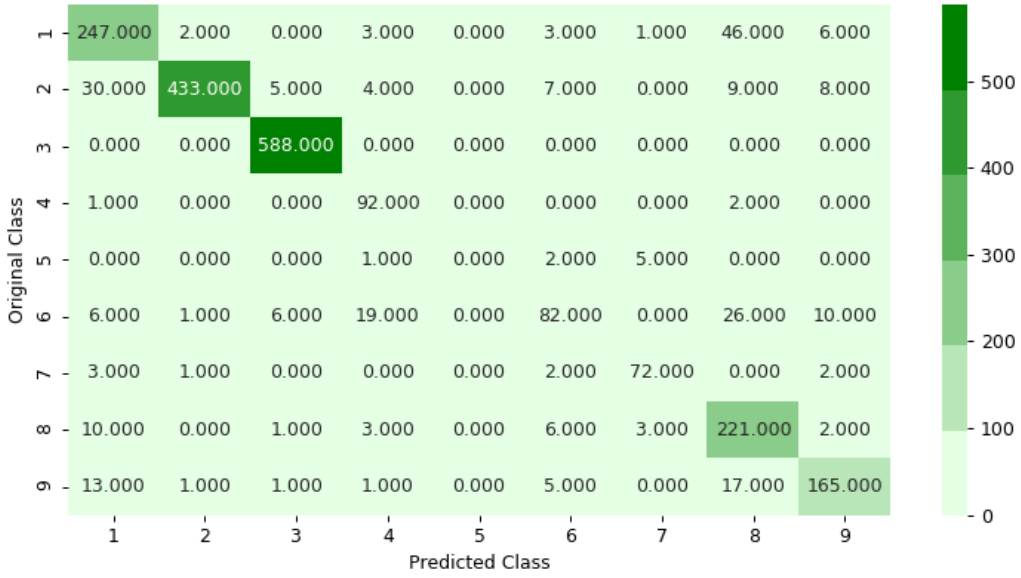
```
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Descent.
# predict(X) Predict class labels for samples in X.

#--------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#--------------------------------

alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=logisticR.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balanced')
logisticR.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train, y_train)
pred_y=sig_clf.predict(X_test)

predict_y = sig_clf.predict_proba(X_train)
print ('log loss for train data',log_loss(y_train, predict_y, labels=logisticR.classes_, eps=1e-15)
)
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logisticR.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_test)
print ('log loss for test data',log_loss(y_test, predict_y, labels=logisticR.classes_, eps=1e-15))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  1e-05 is 1.8861109311791922
log_loss for c =  0.0001 is 1.8845880471197165
log_loss for c =  0.001 is 1.8614285515198798
log_loss for c =  0.01 is 1.437434379095915
log_loss for c =  0.1 is 0.9337959204695321
log_loss for c =  1 is 0.7667190017910965
log_loss for c =  10 is 0.6257185173536978
log_loss for c =  100 is 0.56294262675526
log_loss for c =  1000 is 0.6385231825855628
```

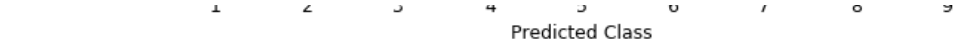




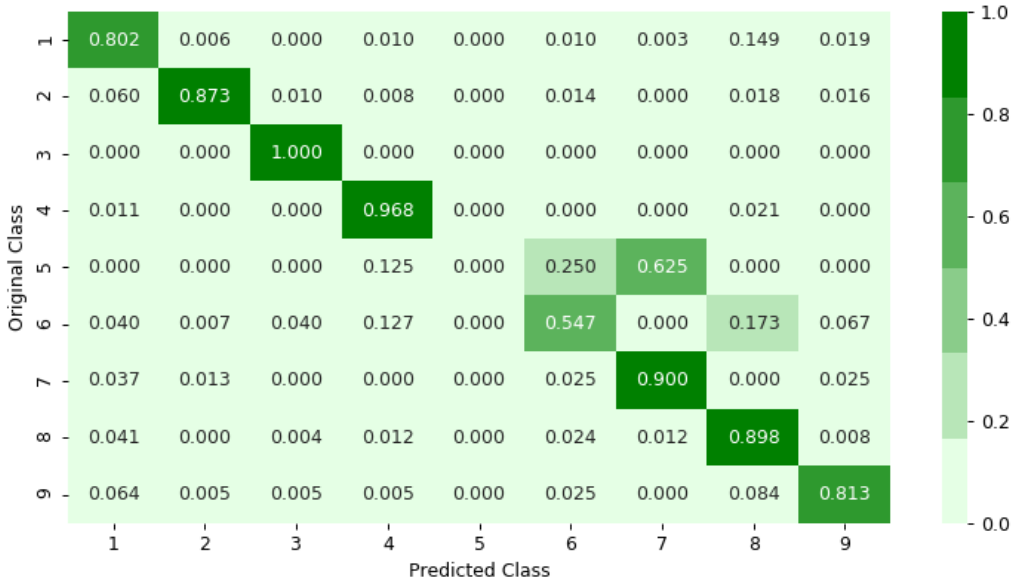

```
log loss for train data 0.4871437301878761
log loss for cv data 0.56294262675526
log loss for test data 0.5294168099375685
Number of misclassified points  12.603495860165593
---------------------------------------------------- Confusion matrix -----------------------------
----------------
```




```
---------------------------------------------------- Precision matrix -----------------------------
----------------
```




```
Sum of columns in precision matrix [ 1.  1.  1.  1. nan  1.  1.  1.  1.]
-------------------------------------------------- Recall matrix ------------------------------------
-------------
```




```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

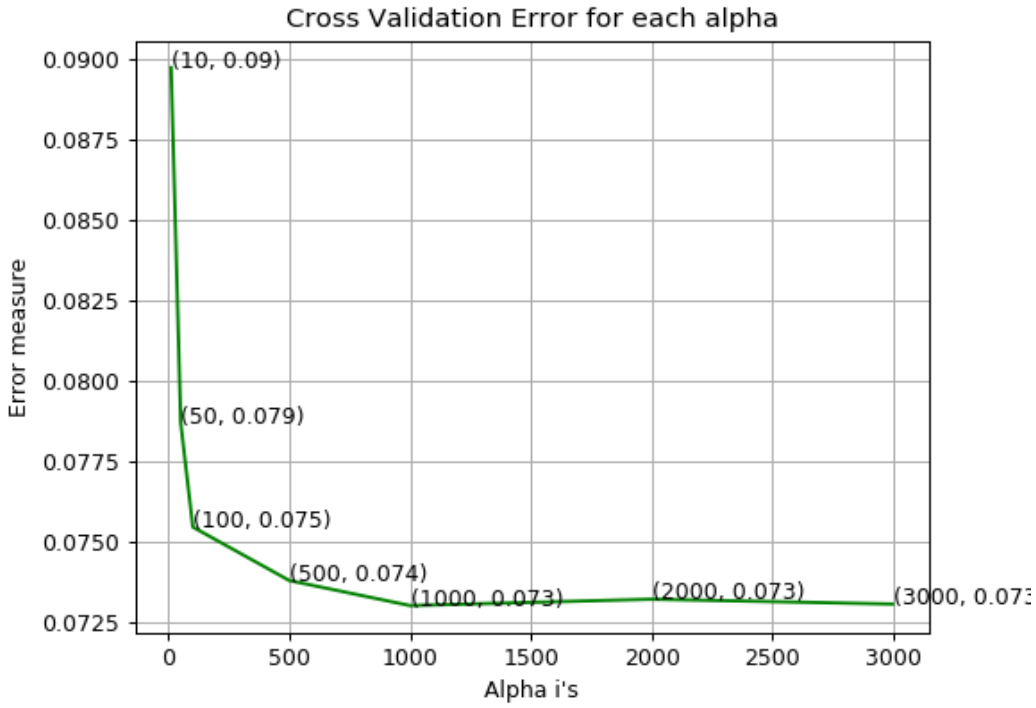
4.1.4. Random Forest Classifier

In [23]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-fores
t-and-their-construction-2/
# --------------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
train_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
```

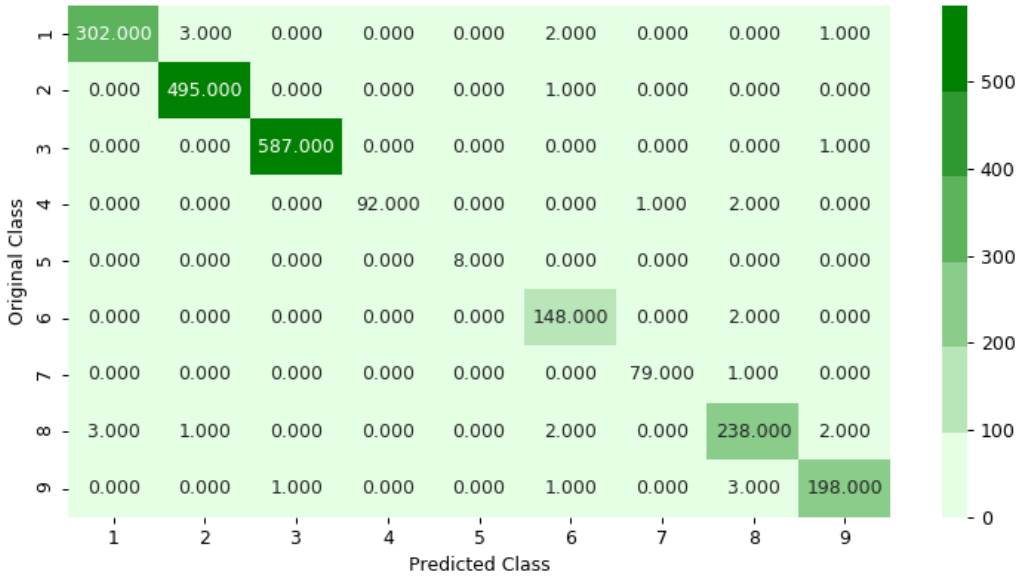
```
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train,
predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y))
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  10 is 0.08975272796356877
log_loss for c =  50 is 0.07869374681057625
log_loss for c =  100 is 0.07546292664044586
log_loss for c =  500 is 0.07379883728342362
log_loss for c =  1000 is 0.07302077078516724
log_loss for c =  2000 is 0.07321813574020479
log_loss for c =  3000 is 0.073068132864414
```

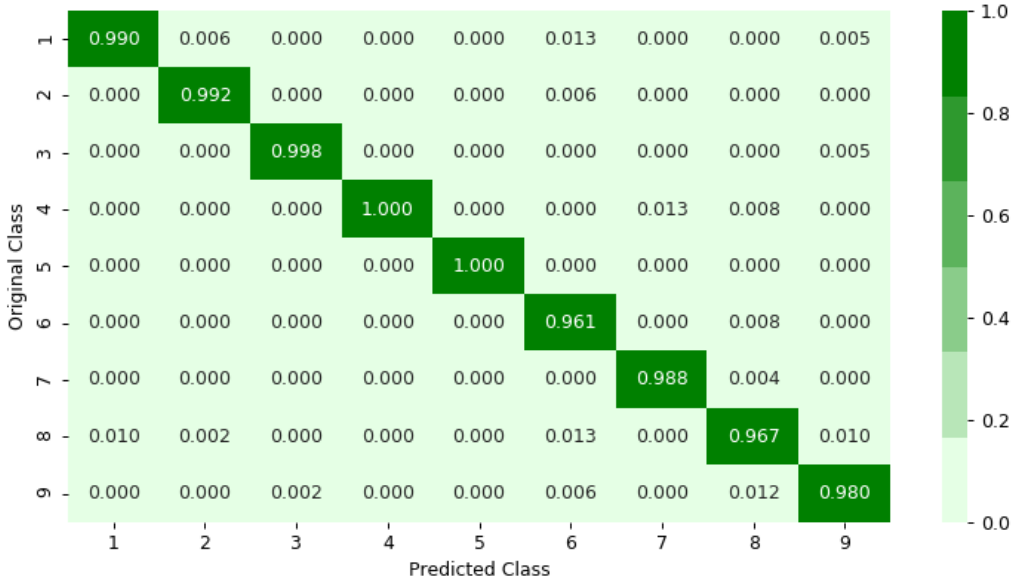



```
For values of best alpha =  1000 The train log loss is: 0.02941015229514485
For values of best alpha =  1000 The cross validation log loss is: 0.07302077078516724
For values of best alpha =  1000 The test log loss is: 0.06623869322452512
```

```
for values of best alpha =  1000 the test log loss is: 0.06623869322452512
Number of misclassified points  1.2419503219871204
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```




```
-------------------------------------------------- Precision matrix ------------------------------
----------------
```




```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
-------------------------------------------------- Recall matrix ---------------------------------
-------------
```

```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

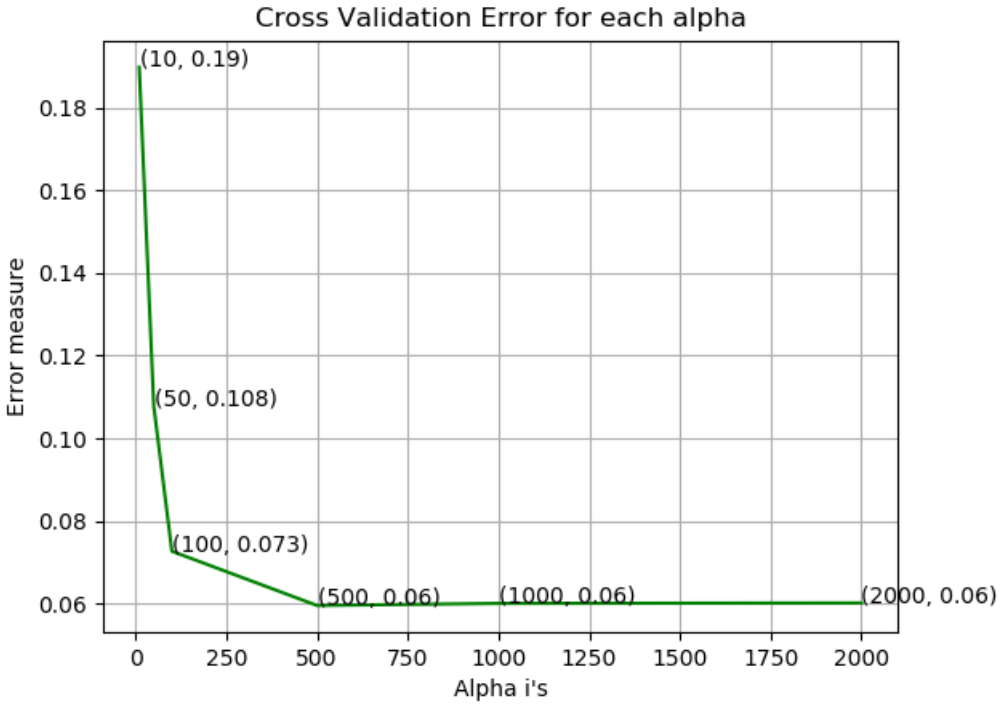
4.1.5. XgBoost Classification

In [14]:

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data
```

```
# find more about XGBClassifier function here
http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGBClassifier
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0,
min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0,
reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbo
se=True, xgb_model=None)
# get_params([deep]) Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is no
t thread safe.
# get_score(importance_type='weight') -> get the feature importance
# ------------------------
# video link1: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/regression-
using-decision-trees-2/
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-en
sembles/
# ------------------------

alpha=[10,50,100,500,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=x_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1)
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train
, predict_y))
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv, predict_y))
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, p
redict_y))
```

```
log_loss for c =  10 is 0.18972194520294353
log_loss for c =  50 is 0.10788109266220497
log_loss for c =  100 is 0.0727450829972164
log_loss for c =  500 is 0.059635927131908094
log_loss for c =  1000 is 0.06014538270053144
log_loss for c =  2000 is 0.060249389062255305
```



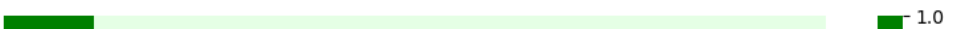

```
For values of best alpha =  500 The train log loss is: 0.02468009568654092
For values of best alpha =  500 The cross validation log loss is: 0.059635927131908094
For values of best alpha =  500 The test log loss is: 0.07847700799402009
```

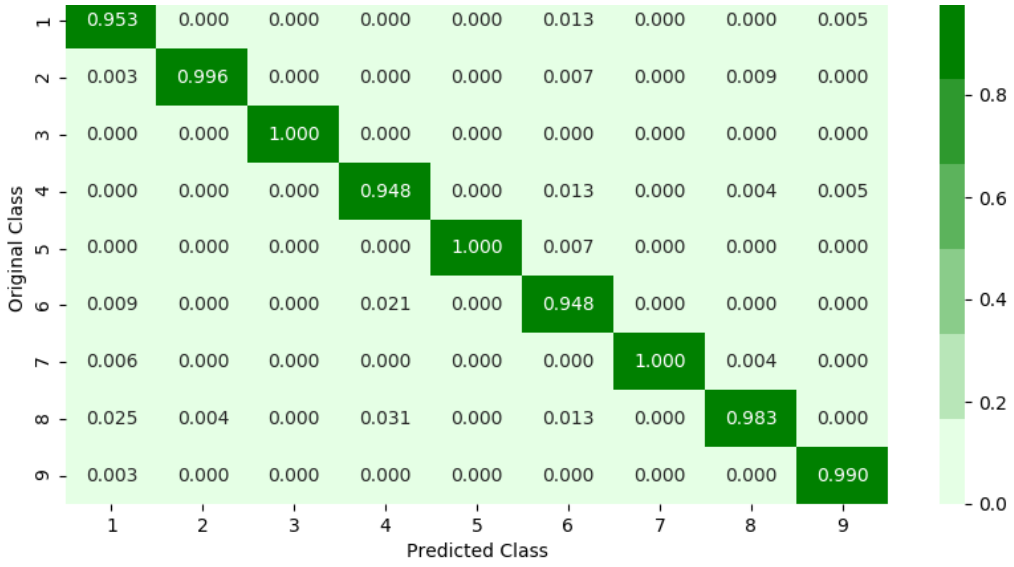
In [15]:

```
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

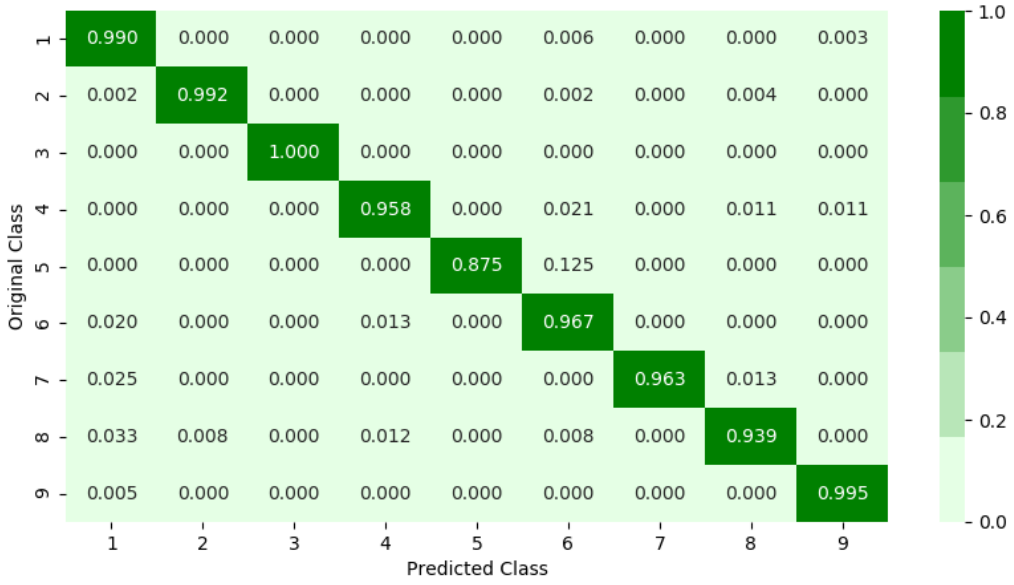
```
Number of misclassified points  1.6559337626494939
---------------------------------------------------- Confusion matrix -----------------------------------
----------------
```


```
---------------------------------------------------- Precision matrix -----------------------------------
----------------
```






```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
---------------------------------------------- Recall matrix ----------------------------------
-------------
```




```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.1.5. XgBoost Classification with best hyper parameters using RandomSearch

In [22]:

```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl1=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl1.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  4.1min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  9.8min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed: 31.8min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 48.9min remaining:  5.4min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 54.2min finished
```

Out[22]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators':
[100, 200, 500, 1000, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'sub
sample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

In [ ]:

```
print (random_cfl1.best_params_)
```

```
{'subsample': 1, 'n_estimators': 2000, 'max_depth': 5, 'learning_rate': 0.01, 'colsample_bytree':
0.5}
```

In [16]:

```
x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.01, colsample_bytree=0.5, max_depth=5,subsam
ple=1)
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

predict_y = c_cfl.predict_proba(X_train)
print ('train loss',log_loss(y_train, predict_y))
predict_y = c_cfl.predict_proba(X_cv)
print ('cv loss',log_loss(y_cv, predict_y))
predict_y = c_cfl.predict_proba(X_test)
print ('test loss',log_loss(y_test, predict_y))
```

```
train loss 0.024348854759529453
cv loss 0.06210692336009718
test loss 0.07717065282799755
```

In [ ]:

```

```

4.2 Modeling with .asm files

There are 10868 files of asm
All the files make up about 150 GB
The asm files contains :
1. Address
2. Segments
3. Opcodes
4. Registers
5. function calls
6. APIs
With the help of parallel processing we extracted all the features In parallel we can use a

With the help of parallel processing we extracted all the features.In parallel we can use a
ll the cores that are present in our computer.

Here we extracted 52 features from all the asm files which are important.

We read the top solutions and handpicked the features from those papers/videos/blogs.
Refer:https://www.kaggle.com/c/malware-classification/discussion

4.2.1 Feature extraction from asm files

- To extract the unigram features from the .asm files we need to process ~150GB of data
- **Note: Below two cells will take lot of time (over 48 hours to complete)**
- We will provide you the output file of these two cells, which you can directly use it

In [ ]:

```
#intially create five folders
#first
#second
#thrid
#fourth
#fifth
#this code tells us about random split of files into five folders
folder_1 ='first'
folder_2 ='second'
folder_3 ='third'
folder_4 ='fourth'
folder_5 ='fifth'
folder_6 = 'output'
for i in [folder_1,folder_2,folder_3,folder_4,folder_5,folder_6]:
    if not os.path.isdir(i):
        os.makedirs(i)

source='train/'
files = os.listdir('train')
ID=df['Id'].tolist()
data=range(0,10868)
r.shuffle(data)
count=0
for i in range(0,10868):
    if i % 5==0:
        shutil.move(source+files[data[i]],'first')
    elif i%5==1:
        shutil.move(source+files[data[i]],'second')
    elif i%5 ==2:
        shutil.move(source+files[data[i]],'thrid')
    elif i%5 ==3:
        shutil.move(source+files[data[i]],'fourth')
    elif i%5==4:
        shutil.move(source+files[data[i]],'fifth')
```

In [ ]:

```
#http://flint.cs.yale.edu/cs421/papers/x86-asm/asm.html

def firstprocess():
    #The prefixes tells about the segments that are present in the asm files
    #There are 450 segments(approx) present in all asm files.
    #this prefixes are best segments that gives us best values.
    #https://en.wikipedia.org/wiki/Data_segment

    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:',
'.tls:','.reloc:','.BSS:','.CODE']
    #this are opcodes that are used to get best results
    #https://en.wikipedia.org/wiki/X86_instruction_listings

    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec',
'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movz
x']
    #best keywords that are taken from different blogs
```

```
    keywords = ['.dll','std::',':dword']
    #Below taken registers are general purpose registers and special registers
    #All the registers which are taken are best
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\asmsmallfile.txt","w+")
    files = os.listdir('first')
    for f in files:
        #filling the values with zeros into the arrays
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        # https://docs.python.org/3/library/codecs.html#codecs.ignore_errors
        # https://docs.python.org/3/library/codecs.html#codecs.Codec.encode
        with codecs.open('first/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                # https://www.tutorialspoint.com/python3/string_rstrip.htm
                line=lines.rstrip().split()
                l=line[0]
                #counting the prefixs in each and every line
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                #counting the opcodes in each and every line
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                #counting registers in the line
                for i in range(len(registers)):
                    for li in line:
                        # we will use registers only in 'text' and 'CODE' segments
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                #counting keywords in the line
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        #pushing the values into the file after reading whole file
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


#same as above
def secondprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:',
'.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec',
'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movz
x']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\mediumasmfile.txt","w+")
    files = os.listdir('second')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
```

```
        with codecs.open('second/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()

# same as smallprocess() functions
def thirdprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:',
'.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec',
'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movz
x']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\largeasmfile.txt","w+")
    files = os.listdir('thrid')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('thrid/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
```

```
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


def fourthprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:',
'.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec',
'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movz
x']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\hugeasmfile.txt","w+")
    files = os.listdir('fourth/')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('fourth/'+f,encoding='cp1252',errors ='replace') as fli:
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


def fifthprocess():
    prefixes = ['HEADER:','.text:','.Pav:','.idata:','.data:','.bss:','.rdata:','.edata:','.rsrc:',
'.tls:','.reloc:','.BSS:','.CODE']
    opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec',
'add','imul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movz
x']
    keywords = ['.dll','std::',':dword']
    registers=['edx','esi','eax','ebx','ecx','edi','ebp','esp','eip']
    file1=open("output\trainasmfile.txt","w+")
    files = os.listdir('fifth/')
    for f in files:
        prefixescount=np.zeros(len(prefixes),dtype=int)
        opcodescount=np.zeros(len(opcodes),dtype=int)
        keywordcount=np.zeros(len(keywords),dtype=int)
        registerscount=np.zeros(len(registers),dtype=int)
        features=[]
        f2=f.split('.')[0]
        file1.write(f2+",")
        opcodefile.write(f2+" ")
        with codecs.open('fifth/'+f,encoding='cp1252',errors ='replace') as fli:
```

```
            for lines in fli:
                line=lines.rstrip().split()
                l=line[0]
                for i in range(len(prefixes)):
                    if prefixes[i] in line[0]:
                        prefixescount[i]+=1
                line=line[1:]
                for i in range(len(opcodes)):
                    if any(opcodes[i]==li for li in line):
                        features.append(opcodes[i])
                        opcodescount[i]+=1
                for i in range(len(registers)):
                    for li in line:
                        if registers[i] in li and ('text' in l or 'CODE' in l):
                            registerscount[i]+=1
                for i in range(len(keywords)):
                    for li in line:
                        if keywords[i] in li:
                            keywordcount[i]+=1
        for prefix in prefixescount:
            file1.write(str(prefix)+",")
        for opcode in opcodescount:
            file1.write(str(opcode)+",")
        for register in registerscount:
            file1.write(str(register)+",")
        for key in keywordcount:
            file1.write(str(key)+",")
        file1.write("\n")
    file1.close()


def main():
    #the below code is used for multiprogramming
    #the number of process depends upon the number of cores present System
    #process is used to call multiprogramming
    manager=multiprocessing.Manager()
    p1=Process(target=firstprocess)
    p2=Process(target=secondprocess)
    p3=Process(target=thirdprocess)
    p4=Process(target=fourthprocess)
    p5=Process(target=fifthprocess)
    #p1.start() is used to start the thread execution
    p1.start()
    p2.start()
    p3.start()
    p4.start()
    p5.start()
    #After completion all the threads are joined
    p1.join()
    p2.join()
    p3.join()
    p4.join()
    p5.join()

if __name__=="__main__":
    main()
```

In [14]:

```
# asmoutputfile.csv(output genarated from the above two cells) will contain all the extracted feat
ures from .asm files
# this file will be uploaded in the drive, you can directly use this
dfasm=pd.read_csv("./Microsoft Malware Detection/asmoutputfile.csv")
Y.columns = ['ID', 'Class']
result_asm = pd.merge(dfasm, Y,on='ID', how='left')
result_asm.head()
```

Out[14]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	edx	esi	eax	ebx	ecx	edi	eb
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	18	66	15	43	83	0	1
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	18	29	48	82	12	0	1
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	13	42	10	67	14	0	1

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	edx	esi	eax	ebx	ecx	edi	eb
3	3X2nY7iQaPBlWDrAZqJe	17	227	0	43	19	0	0	0	3	...	6	8	14	7	2	0	
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	12	9	18	29	5	0	1

5 rows × 53 columns

4.2.1.1 Files sizes of each .asm file

In [16]:

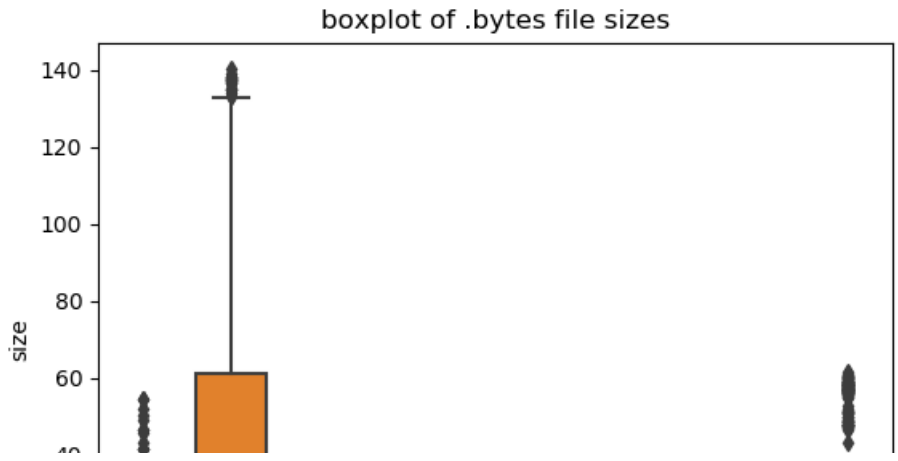
```
#file sizes of byte files

files=os.listdir('./Microsoft Malware Detection/asmFiles')
filenames=Y['ID'].tolist()
class_y=Y['Class'].tolist()
class_bytes=[]
sizebytes=[]
fnames=[]
for file in files:
    # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
    # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_nlink=1,
st_uid=0, st_gid=0,
    # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519638522)
    # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.htm
    statinfo=os.stat('./Microsoft Malware Detection/asmFiles/'+file)
    # split the file name at '.' and take the first part of it i.e the file name
    file=file.split('.')[0]
    if any(file == filename for filename in filenames):
        i=filenames.index(file)
        class_bytes.append(class_y[i])
        # converting into Mb's
        sizebytes.append(statinfo.st_size/(1024.0*1024.0))
        fnames.append(file)
asm_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
print (asm_size_byte.head())
```

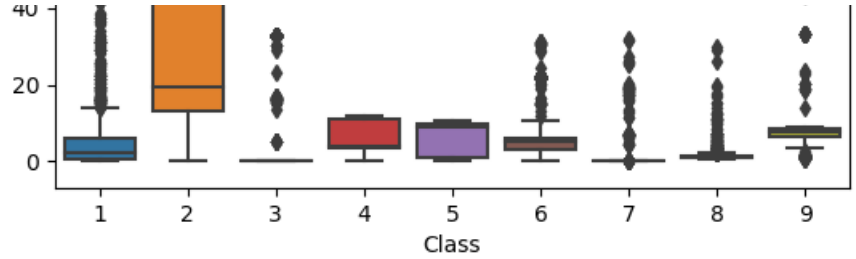
```
   Class                       ID       size
0      1  aMSX7lvrZeCBwE9kUnDf   1.011998
1      3  epj8anqLcrRxHGVWT1Bg  32.695071
2      3  3rt2gI9JVaZcTQGyjmbk  32.701280
3      3  aYRoe1Nvt4J8Bz6LbSZ5   0.163288
4      4  gIONM8E513LYstAjyzWl  11.216479
```

4.2.1.2 Distribution of .asm file sizes

In [15]:

```
#boxplot of asm files
ax = sns.boxplot(x="Class", y="size", data=asm_size_byte)
plt.title("boxplot of .bytes file sizes")
plt.show()
```





In [17]:

```
# add the file size feature to previous extracted features
print(result_asm.shape)
print(asm_size_byte.shape)
result_asm = pd.merge(result_asm, asm_size_byte.drop(['Class'], axis=1),on='ID', how='left')
result_asm.head()
```

```
(10868, 53)
(10868, 3)
```

Out[17]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	ecx	edi	ebp	es
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	66	15	43	83	0	17	4
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	29	48	82	12	0	14	
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	42	10	67	14	0	11	
3	3X2nY7iQaPBlWDrAZqJe	17	227	0	43	19	0	0	0	3	...	8	14	7	2	0	8	
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	9	18	29	5	0	11	

5 rows × 54 columns

In [18]:

```
# we normalize the data each column
result_asm.head()
```

Out[18]:

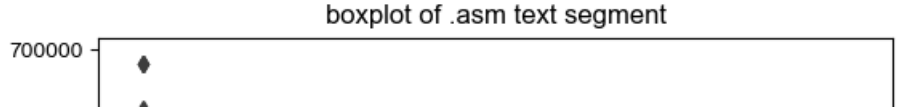
	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	ecx	edi	ebp	es
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	66	15	43	83	0	17	4
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	29	48	82	12	0	14	
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	42	10	67	14	0	11	
3	3X2nY7iQaPBlWDrAZqJe	17	227	0	43	19	0	0	0	3	...	8	14	7	2	0	8	
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	9	18	29	5	0	11	

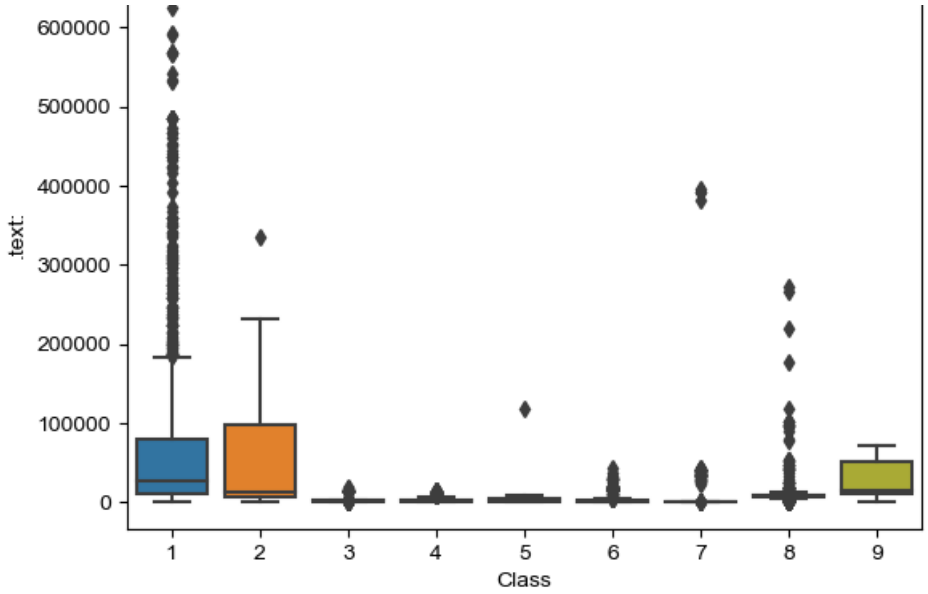
5 rows × 54 columns

4.2.2 Univariate analysis on asm file features

In [18]:

```
ax = sns.boxplot(x="Class", y=".text:", data=result_asm)
plt.title("boxplot of .asm text segment")
plt.show()
```

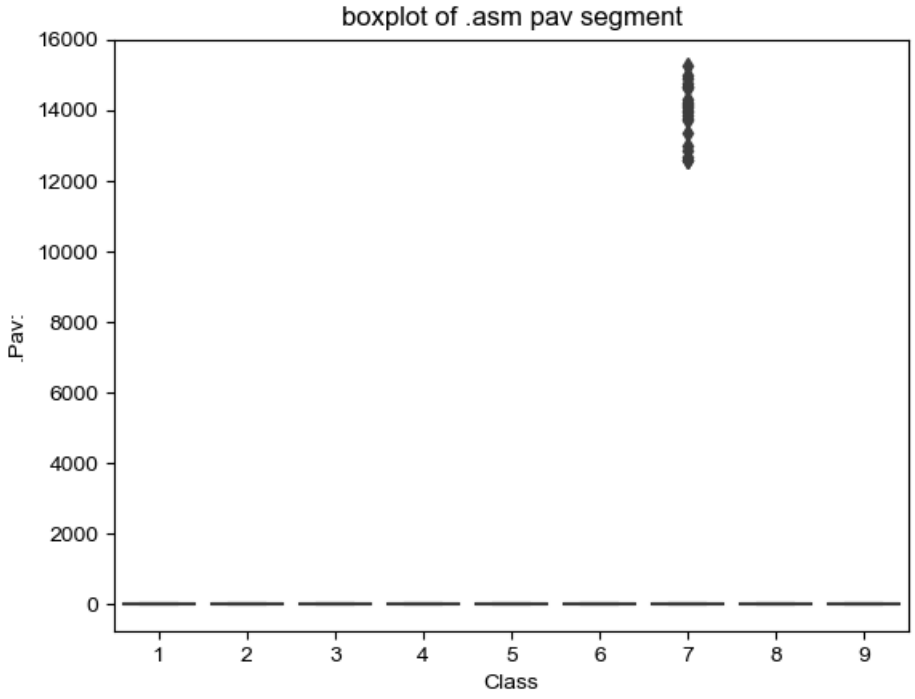




The plot is between Text and class
Class 1,2 and 9 can be easly separated

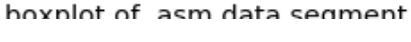
In [19]:

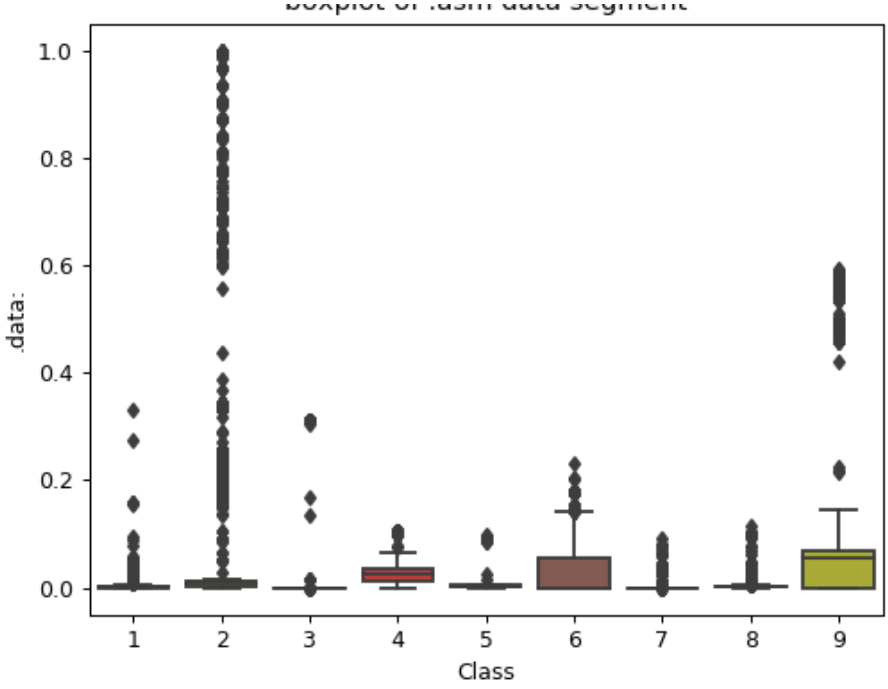
```
ax = sns.boxplot(x="Class", y=".Pav:", data=result_asm)
plt.title("boxplot of .asm pav segment")
plt.show()
```



In [19]:

```
ax = sns.boxplot(x="Class", y=".data:", data=result_asm)
plt.title("boxplot of .asm data segment")
plt.show()
```

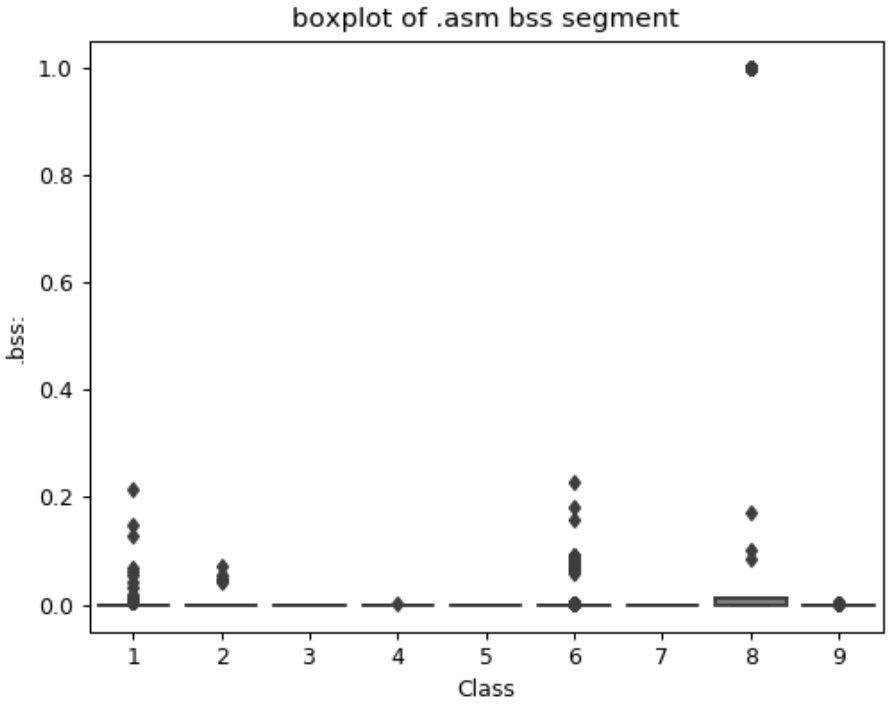




The plot is between data segment and class label
class 6 and class 9 can be easily separated from given points

In [20]:

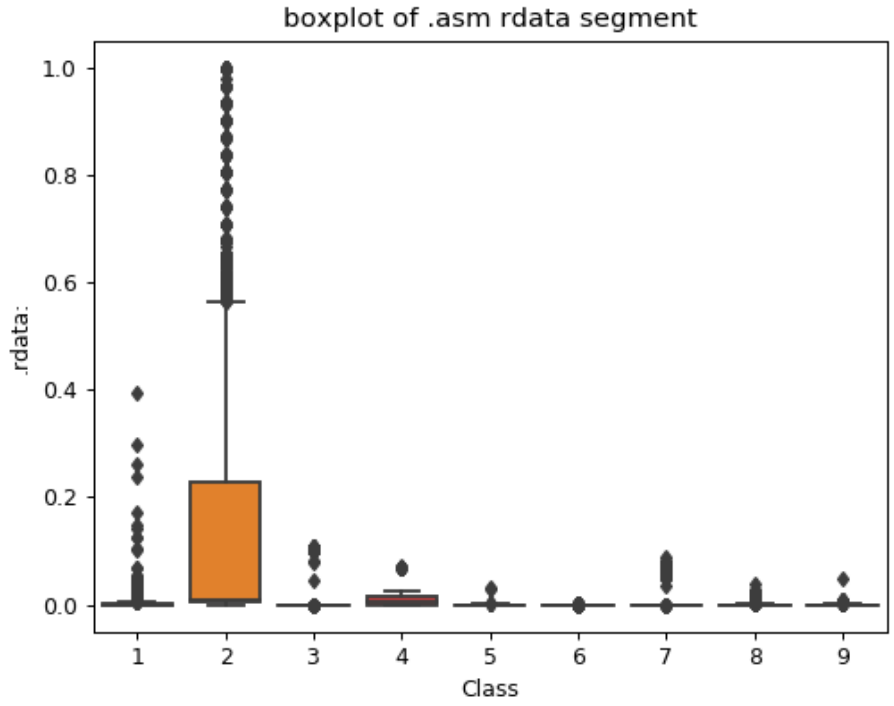
```
ax = sns.boxplot(x="Class", y=".bss:", data=result_asm)
plt.title("boxplot of .asm bss segment")
plt.show()
```



plot between bss segment and class label
very less number of files are having bss segment

In [21]:

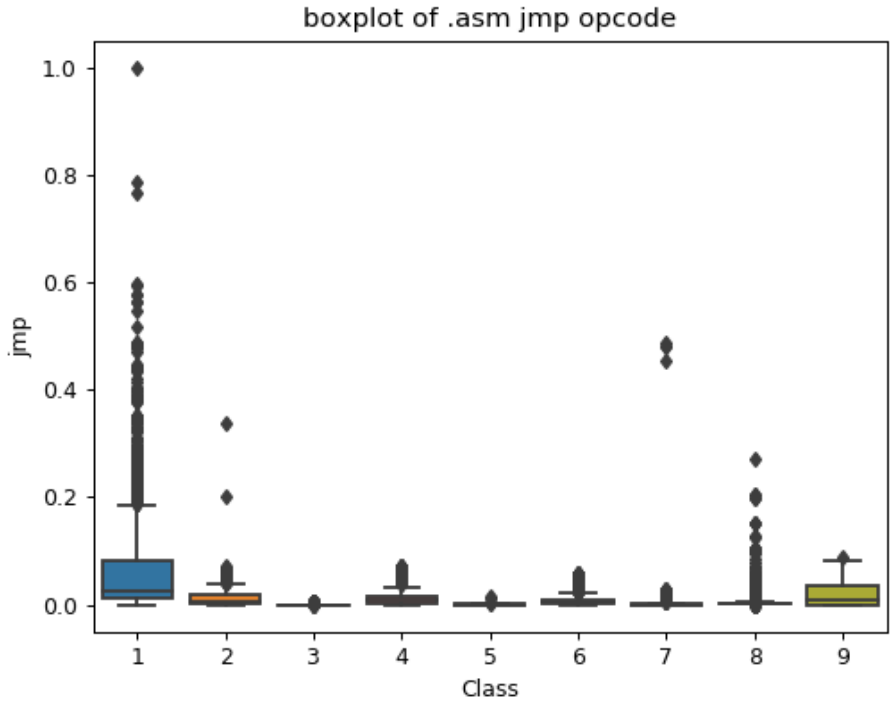
```
ax = sns.boxplot(x="Class", y=".rdata:", data=result_asm)
plt.title("boxplot of .asm rdata segment")
plt.show()
```



Plot between rdata segment and Class segment
Class 2 can be easily separated 75 pecentile files are having 1M rdata lines

In [22]:

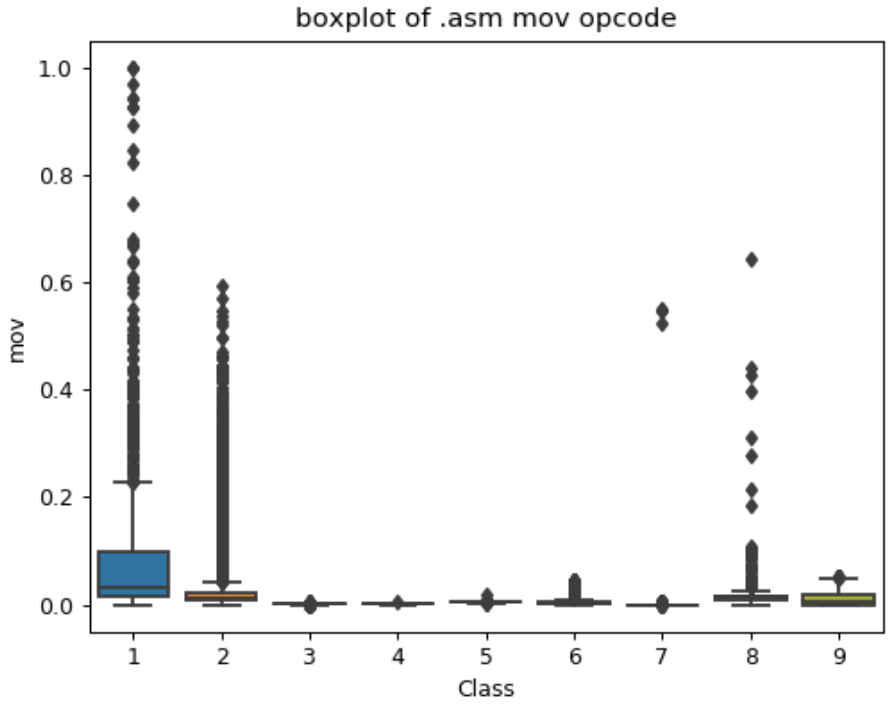
```
ax = sns.boxplot(x="Class", y="jmp", data=result_asm)
plt.title("boxplot of .asm jmp opcode")
plt.show()
```



plot between jmp and Class label
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [23]:

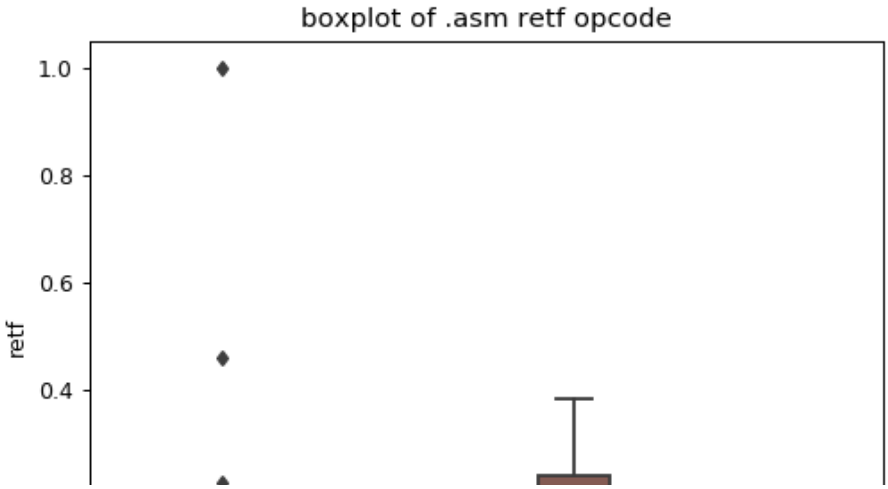
```
ax = sns.boxplot(x="Class", y="mov", data=result_asm)
plt.title("boxplot of .asm mov opcode")
plt.show()
```

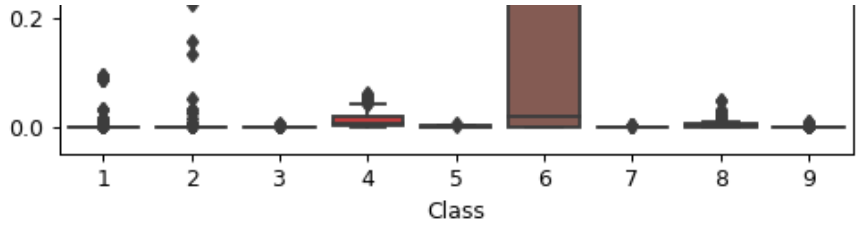


plot between Class label and mov opcode
Class 1 is having frequency of 2000 approx in 75 perentile of files

In [24]:

```
ax = sns.boxplot(x="Class", y="retf", data=result_asm)
plt.title("boxplot of .asm retf opcode")
plt.show()
```

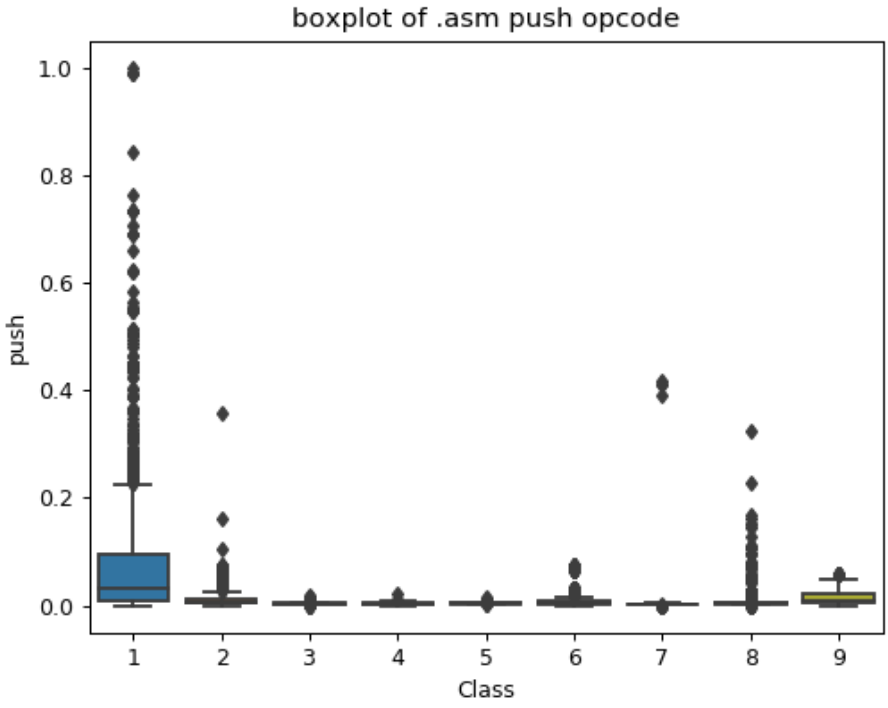




plot between Class label and retf
Class 6 can be easily separated with opcode retf
The frequency of retf is approx of 250.

In [25]:

```
ax = sns.boxplot(x="Class", y="push", data=result_asm)
plt.title("boxplot of .asm push opcode")
plt.show()
```



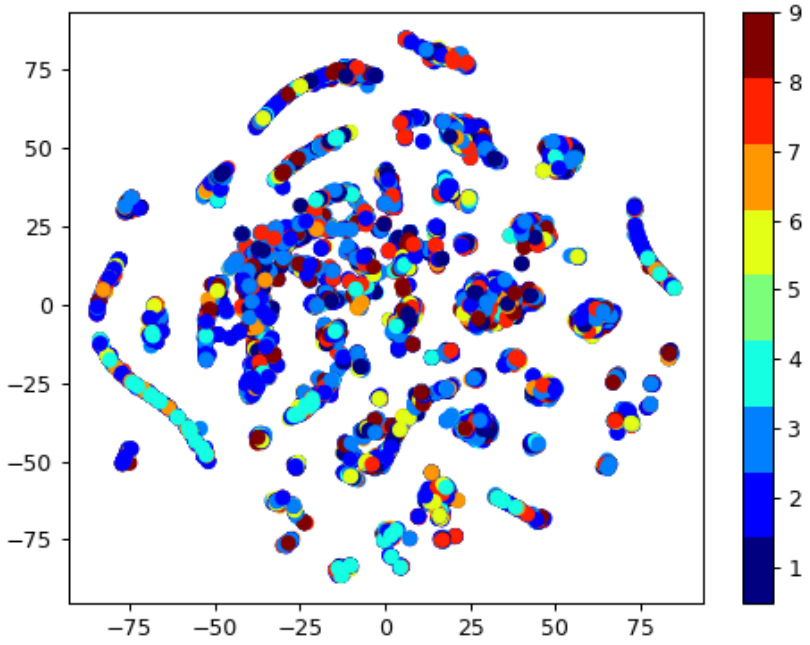
plot between push opcode and Class label
Class 1 is having 75 precentile files with push opcodes of frequency 1000

4.2.2 Multivariate Analysis on .asm file features

In [16]:

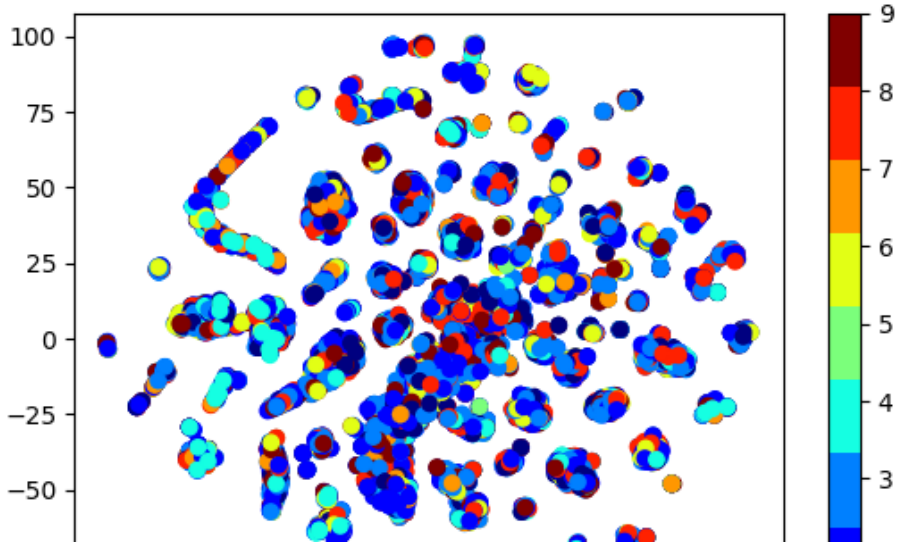
```
# check out the course content for more explantion on tsne algorithm
# https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/t-distributed-stochastic
-neighbourhood-embeddingt-sne-part-1/

#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result_asm.drop(['ID','Class'], axis=1).fillna(0))
vis_x = results[:, 0]
vis_y = results[:, 1   ]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
```

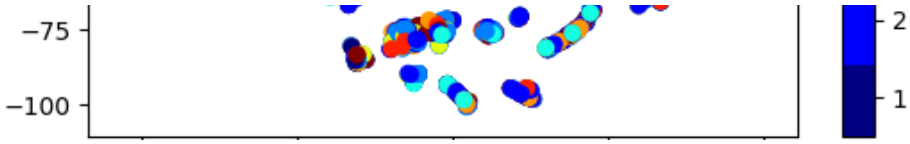
```
plt.clim(0.5, 9)
plt.show()
```



In [30]:

```
# by univariate analysis on the .asm file features we are getting very negligible information from

# 'rtn', '.BSS:' '.CODE' features, so heare we are trying multivariate analysis after removing tho
se features
# the plot looks very messy

xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result_asm.drop(['ID','Class', 'rtn', '.BSS:', '.CODE','size'], axis=1
))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```





TSNE for asm data with perplexity 50

4.2.3 Conclusion on EDA

- We have taken only 52 features from asm files (after reading through many blogs and research papers)
- The univariate analysis was done only on few important features.
- Take-aways
  - 1. Class 3 can be easily separated because of the frequency of segments,opcodes and keywords being less
  - 2. Each feature has its unique importance in separating the Class labels.

4.3 Train and test split

In [18]:

```
asm_y = result_asm['Class']
asm_x = result_asm.drop(['ID','Class','.BSS:','rtn','.CODE'], axis=1)
```

In [19]:

```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,stratify=asm_y,tes
t_size=0.20)
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_train_asm,stratify=y
_train_asm,test_size=0.20)
```

In [20]:

```
print( X_cv_asm.isnull().all())
```

```
HEADER:    False
.text:     False
.Pav:      False
.idata:    False
.data:     False
.bss:      False
.rdata:    False
.edata:    False
.rsrc:     False
.tls:      False
.reloc:    False
jmp        False
mov        False
retf       False
push       False
pop        False
xor        False
retn       False
nop        False
sub        False
inc        False
dec        False
add        False
imul       False
xchg       False
or         False
shr        False
cmp        False
call       False
shl        False
ror        False
rol        False
```

```
jnb         False
jz          False
lea         False
movzx       False
.dll        False
std::       False
:dword      False
edx         False
esi         False
eax         False
ebx         False
ecx         False
edi         False
ebp         False
esp         False
eip         False
size        False
dtype: bool
```

4.4. Machine Learning models on features of .asm files

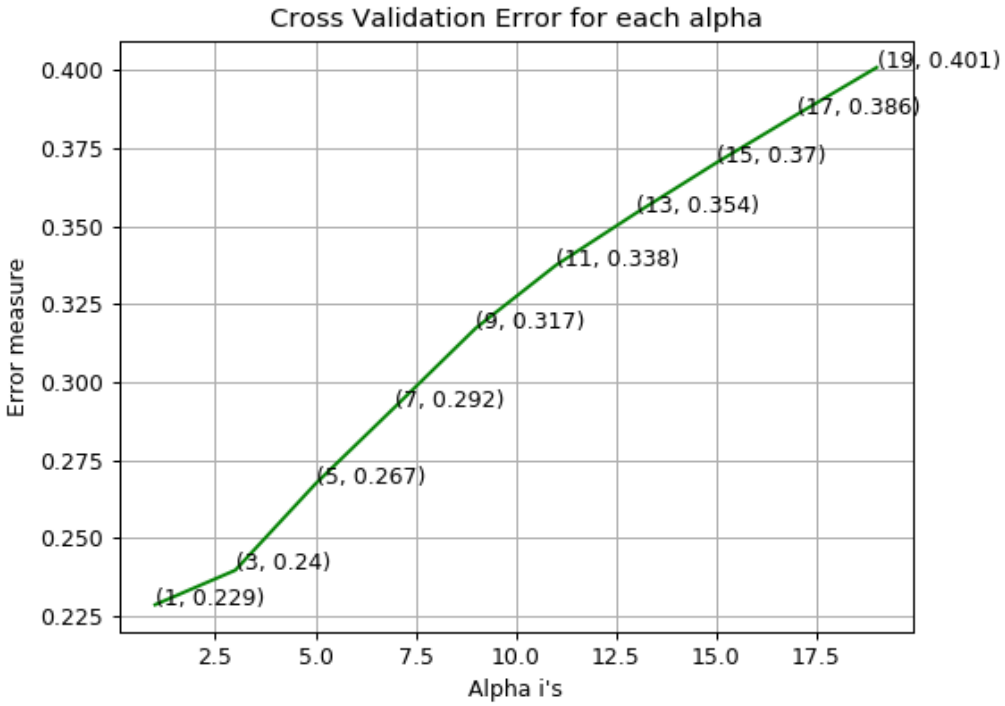
4.4.1 K-Nearest Neigbors

In [35]:

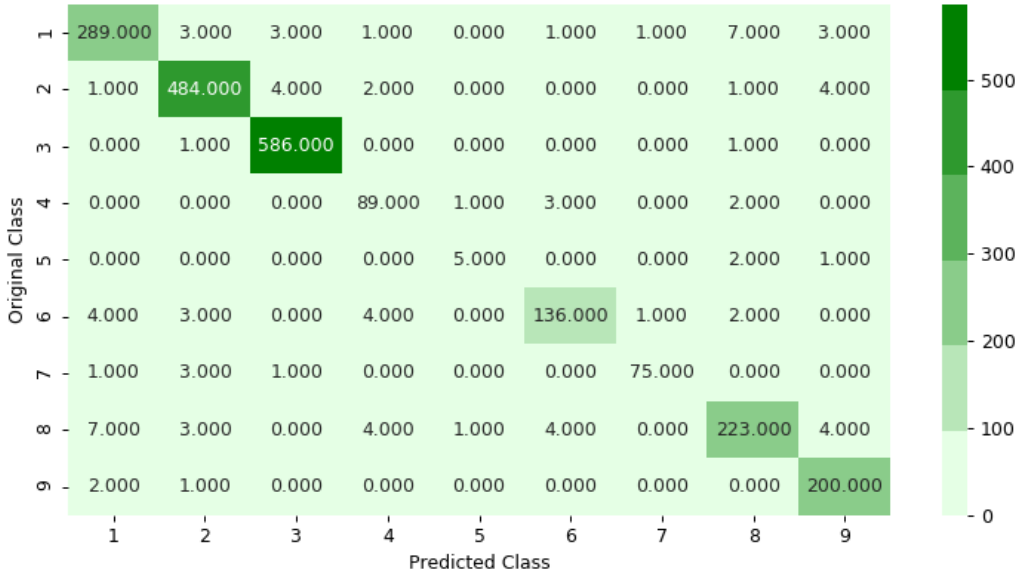
```
# find more about KNeighborsClassifier() here http://scikit-
learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-----------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-ne
ighbors-geometric-intuition-with-a-toy-example-1/
#-----------------------------------


# find more about CalibratedClassifier
#CV here at http://scikit-
learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight]) Fit the calibrated model
# get_params([deep]) Get parameters for this estimator.
# predict(X) Predict the target of new samples.
# predict_proba(X) Posterior probabilities of classification
#-----------------------------------
# video link:
#-----------------------------------

alpha = [x for x in range(1, 21,2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=k_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for k = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)
```

```
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

k_cfl=KNeighborsClassifier(n_neighbors=alpha[best_alpha])
k_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)
pred_y=sig_clf.predict(X_test_asm)


predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',log_loss(y_train_asm, predict_y))
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',log_loss(y_cv_asm, predict_y))
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',log_loss(y_test_asm, predict_y))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

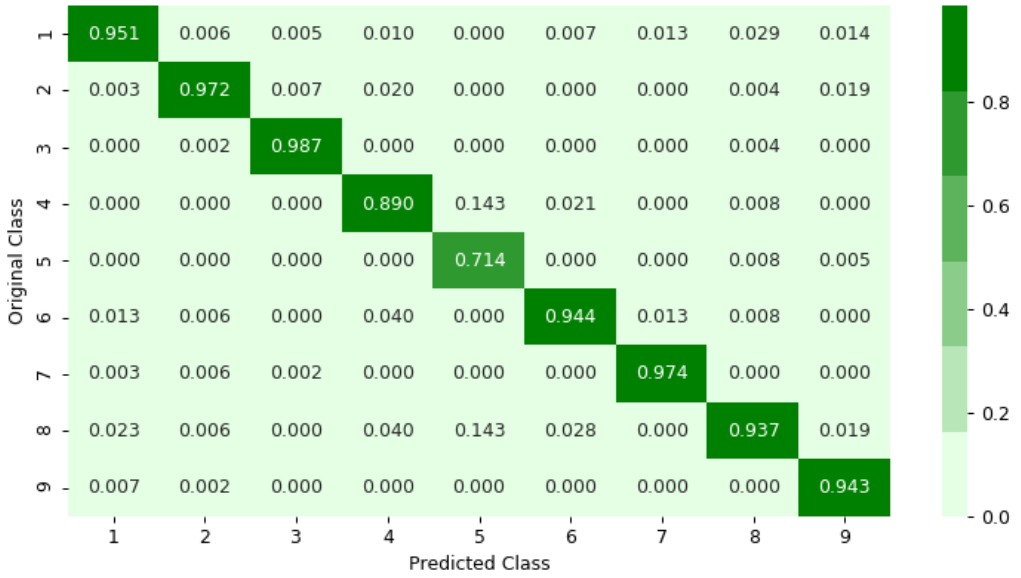
```
log_loss for k =  1 is 0.2286951008264786
log_loss for k =  3 is 0.23974604909767921
log_loss for k =  5 is 0.26743767182569295
log_loss for k =  7 is 0.2922812711849662
log_loss for k =  9 is 0.3173517943176062
log_loss for k =  11 is 0.3375272301343973
log_loss for k =  13 is 0.3542581717184334
log_loss for k =  15 is 0.3703567252351854
log_loss for k =  17 is 0.3857570471590401
log_loss for k =  19 is 0.40090334916939535
```




```
log loss for train data 0.07371820550676085
log loss for cv data 0.2286951008264786
log loss for test data 0.2135354369723588
Number of misclassified points  4.001839926402944
---------------------------------------------------- Confusion matrix -----------------------------
----------------
```

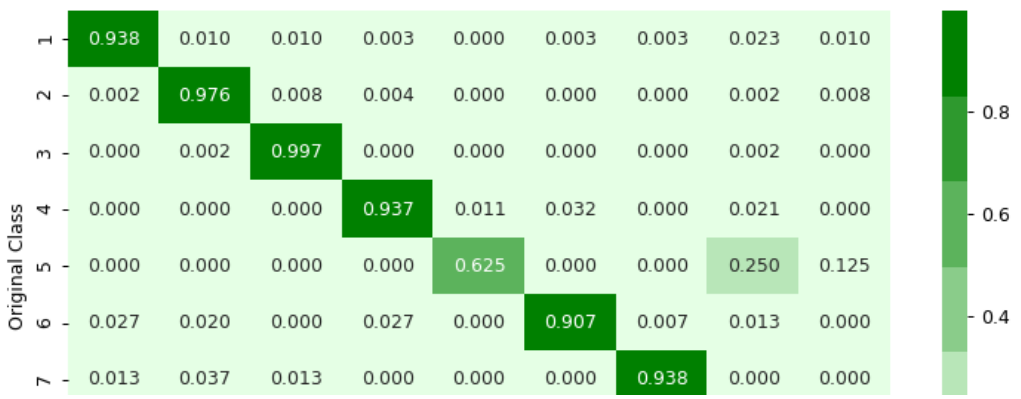


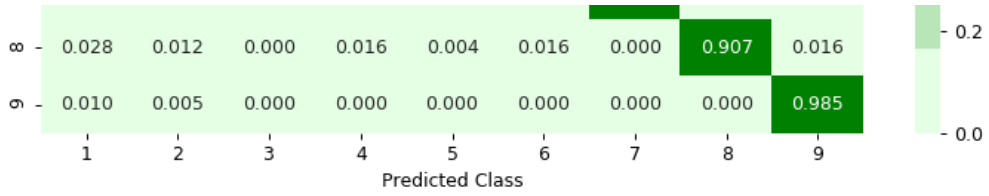
---------- **Precision matrix** ----------



Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]

---------- **Recall matrix** ----------





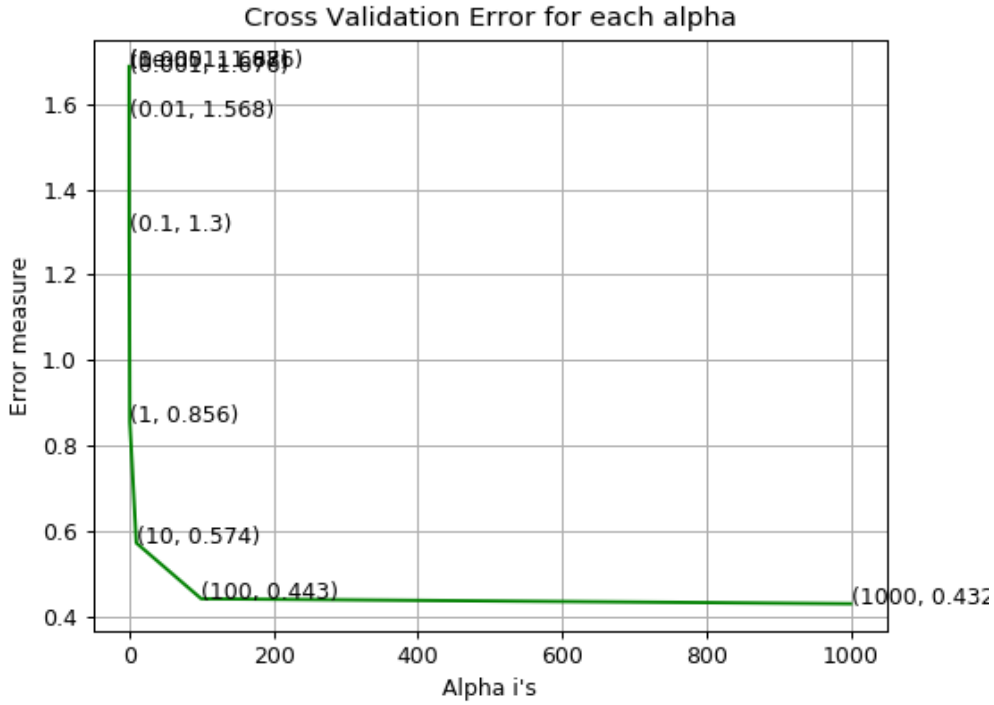
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]

4.4.2 Logistic Regression

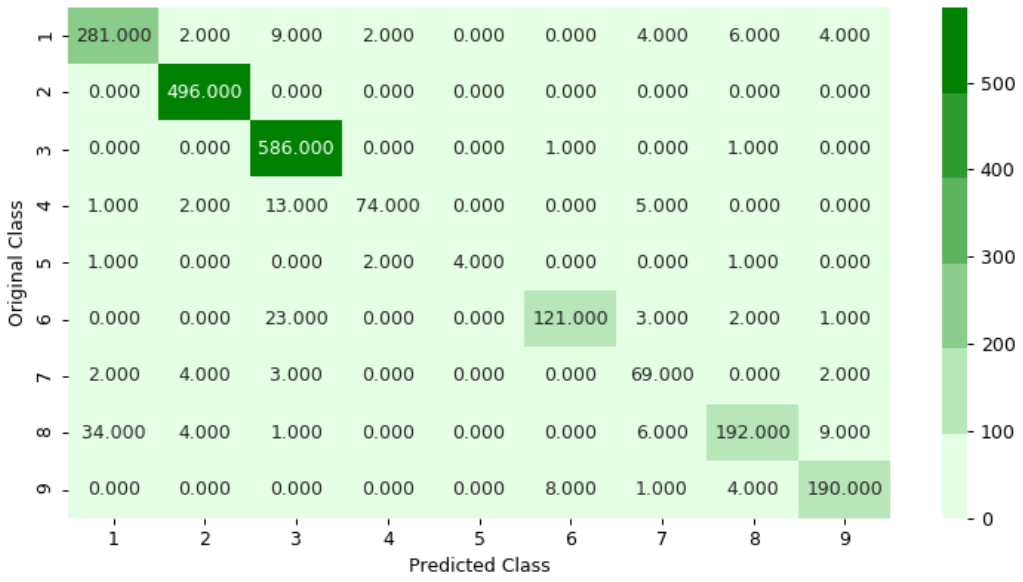
In [36]:

```
# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Descent.
# predict(X) Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#-------------------------------


alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=logisticR.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balanced')
logisticR.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',(log_loss(y_train_asm, predict_y, labels=logisticR.classes_, eps=1
e-15)))
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=logisticR.classes_, eps=1e-15))
)
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=logisticR.classes_, eps=1e-
15)))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

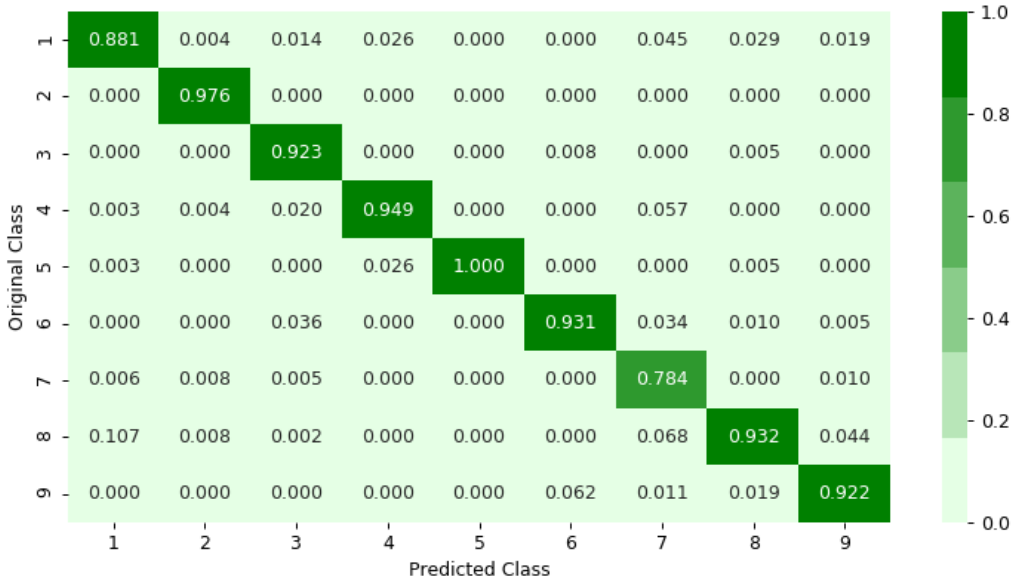
```
log_loss for c =  1e-05 is 1.6869859868957804
log_loss for c =  0.0001 is 1.6855010192472757
log_loss for c =  0.001 is 1.6755781562152487
log_loss for c =  0.01 is 1.5677273322121714
log_loss for c =  0.1 is 1.3002573116338927
log_loss for c =  1 is 0.856048258533692
log_loss for c =  10 is 0.5735687649879864
log_loss for c =  100 is 0.4431214718098947
log_loss for c =  1000 is 0.43157353232283385
```



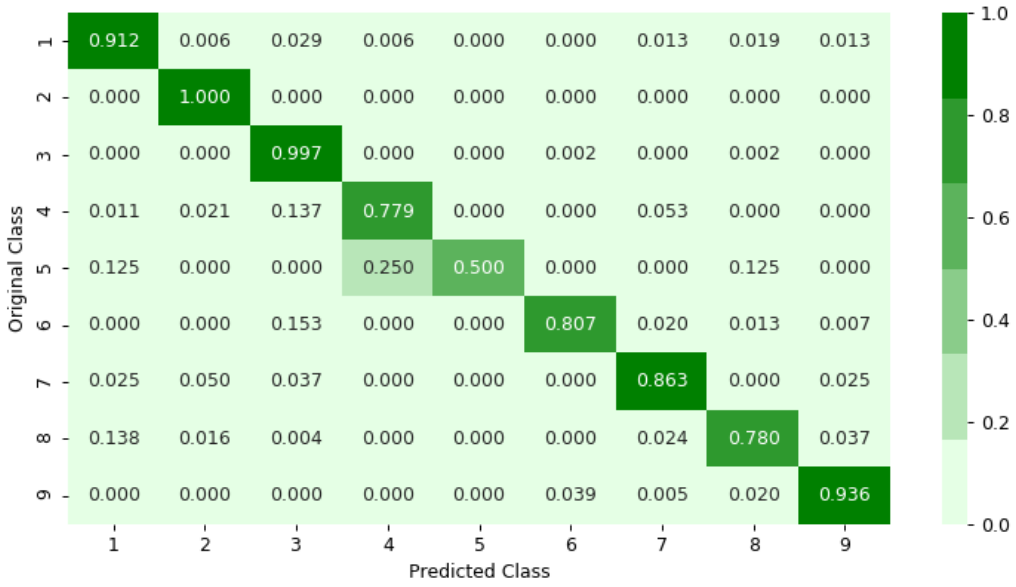

```
log loss for train data 0.38150548196180933
log loss for cv data 0.43157353232283385
log loss for test data 0.38308926013384326
Number of misclassified points  7.405703771849126
---------------------------------------------------- Confusion matrix -----------------------------
----------------
```



```
-------------------------------------------------- Precision matrix ---------------------------------
---------------
```




```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
-------------------------------------------------- Recall matrix ------------------------------------
-------------
```




```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

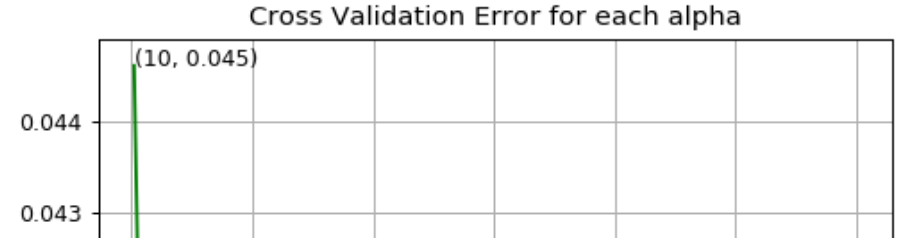
4.4.3 Random Forest Classifier

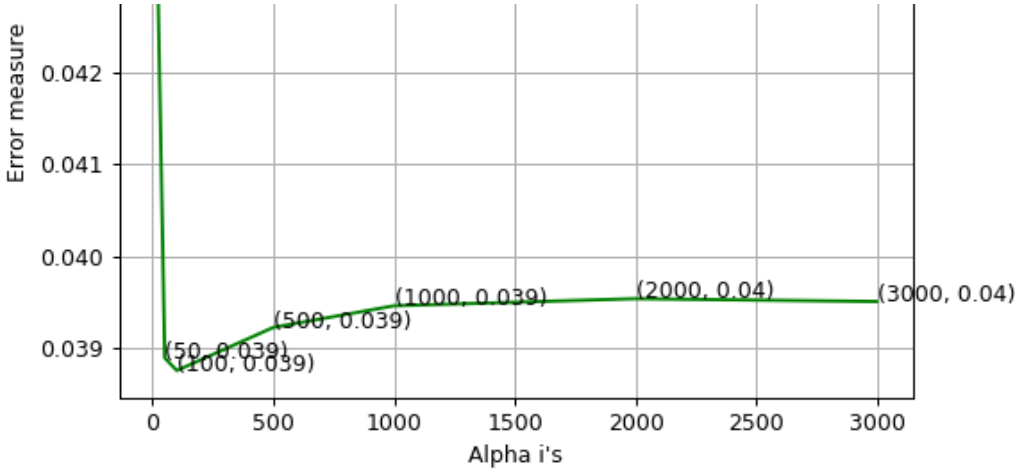
In [37]:

```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False
```

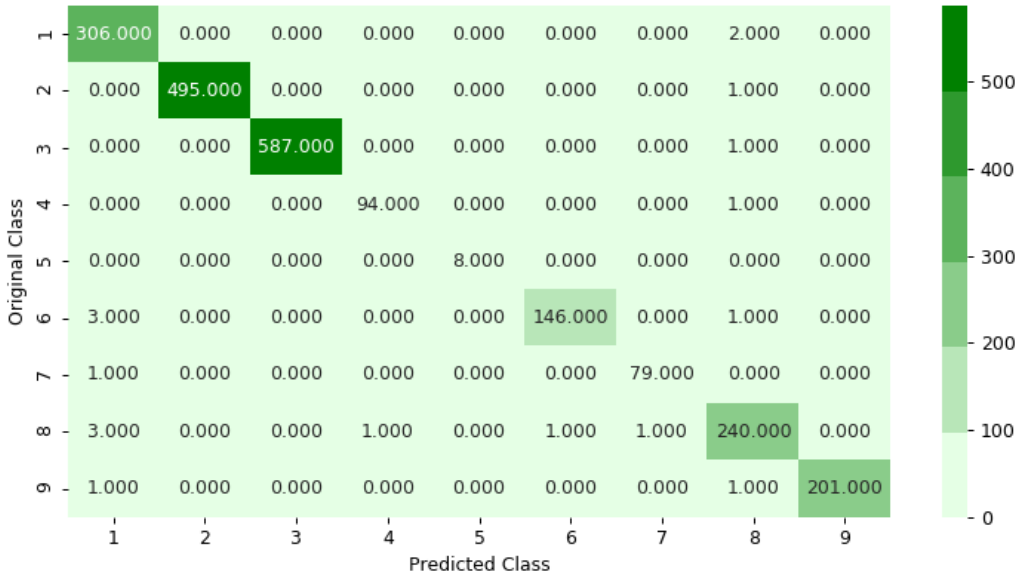
```
verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# --------------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=r_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)
predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',(log_loss(y_train_asm, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=sig_clf.classes_, eps=1e-15)))
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=sig_clf.classes_, eps=1e-15)))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
log_loss for c =  10 is 0.044604000720488056
log_loss for c =  50 is 0.038892329851189296
log_loss for c =  100 is 0.03875524544813011
log_loss for c =  500 is 0.039224440805809314
log_loss for c =  1000 is 0.03945941790783839
log_loss for c =  2000 is 0.03953659123286974
log_loss for c =  3000 is 0.03950608587732239
```

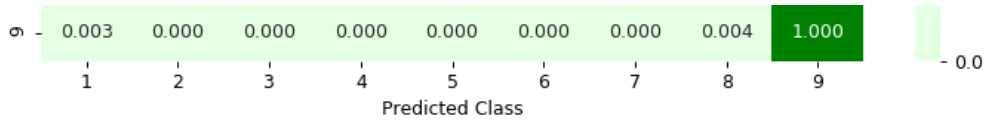




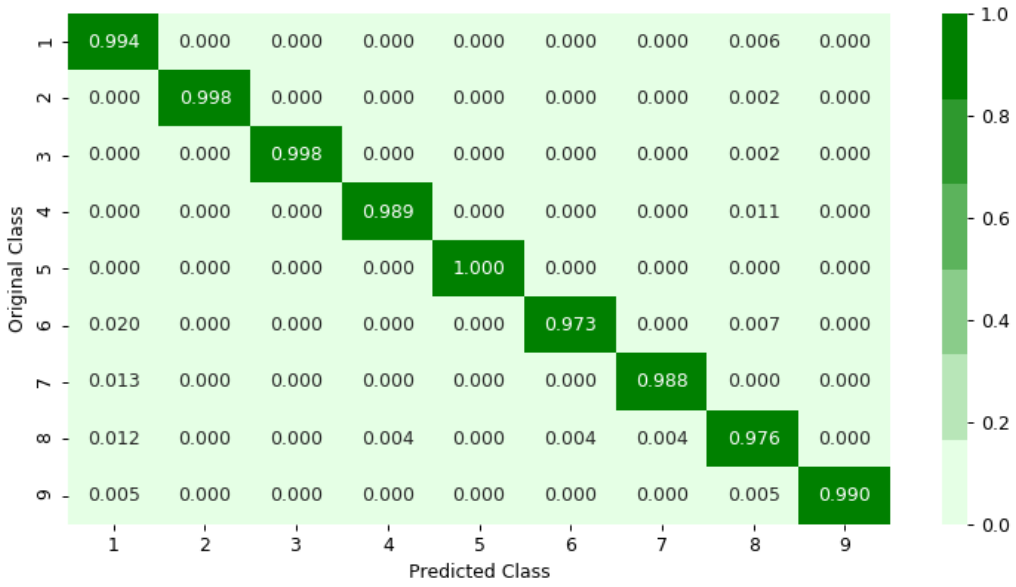

```
log loss for train data 0.012379247850927044
log loss for cv data 0.03875524544813011
log loss for test data 0.039428378936875376
Number of misclassified points  0.8279668813247469
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```




```
-------------------------------------------------- Precision matrix ------------------------------
----------------
```




```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
-------------------------------------------------- Recall matrix ----------------------------------
------------
```




```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

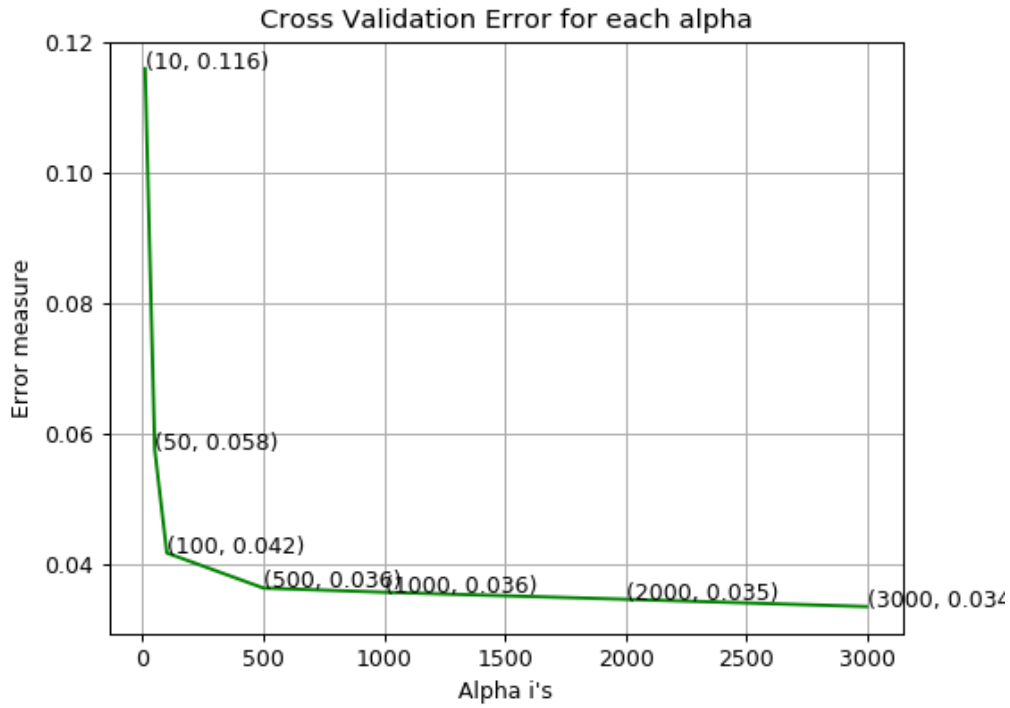
4.4.4 XgBoost Classifier

In [38]:

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here
http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGBClassifier
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0,
min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0,
reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbo
se=True, xgb_model=None)
# get_params([deep]) Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is no
t thread safe.
# get_score(importance_type='weight') -> get the feature importance
# ------------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-en
sembles/
# ------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
```

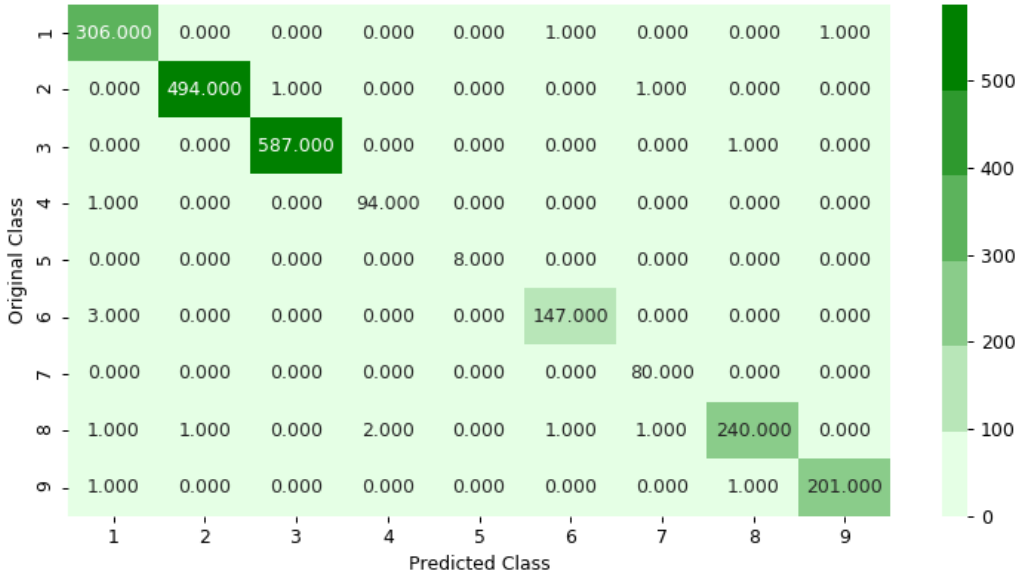
```
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=x_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1)
x_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

predict_y = sig_clf.predict_proba(X_train_asm)

print ('For values of best alpha = ', alpha[best_alpha], "The train log loss
is:",log_loss(y_train_asm, predict_y))
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv_asm, predict_y))
predict_y = sig_clf.predict_proba(X_test_asm)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss
is:",log_loss(y_test_asm, predict_y))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_asm))
```

```
log_loss for c =  10 is 0.11588105338340265
log_loss for c =  50 is 0.057658250882591494
log_loss for c =  100 is 0.04186141305711363
log_loss for c =  500 is 0.03649854125696994
log_loss for c =  1000 is 0.035859619519393905
log_loss for c =  2000 is 0.03478236752207586
log_loss for c =  3000 is 0.033667303437409195
```

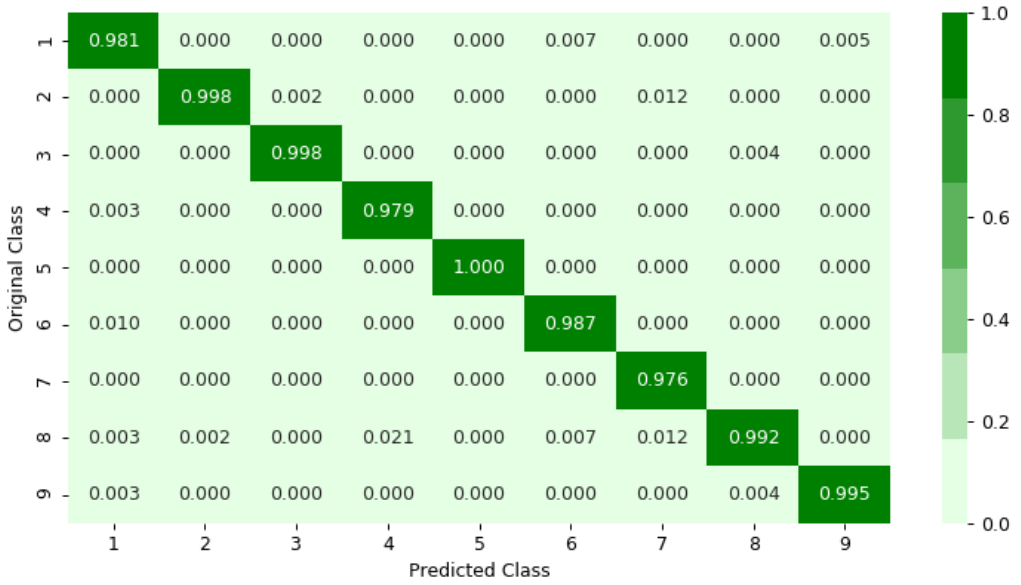



```
For values of best alpha =  3000 The train log loss is: 0.00982726018742022
```

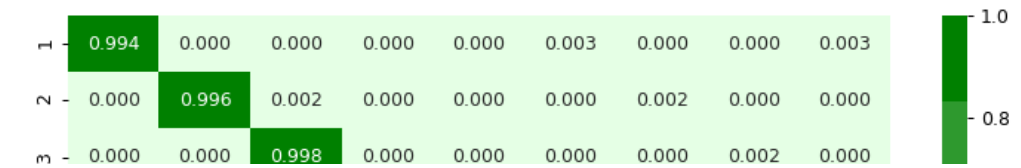
```
For values of best alpha =  3000 The cross validation log loss is: 0.033667303437409195
For values of best alpha =  3000 The test log loss is: 0.042877055973511075
Number of misclassified points  0.78196872125115
-------------------------------------------------- Confusion matrix ------------------------------
----------------
```

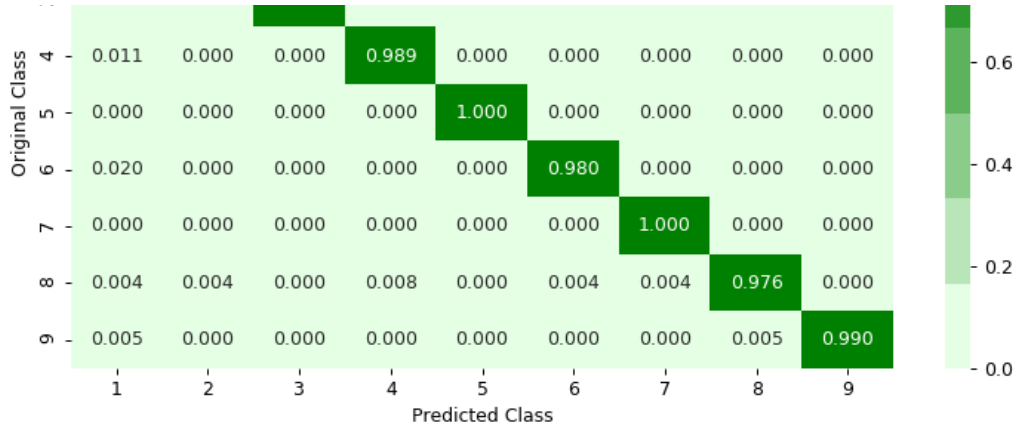



```
-------------------------------------------------- Precision matrix ------------------------------
----------------
```




```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
-------------------------------------------------- Recall matrix ---------------------------------
-------------
```






```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

4.4.5 Xgboost Classifier with best hyperparameters

In [39]:

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train_asm,y_train_asm)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  5.6min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed:  6.4min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:  9.3min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 15.6min remaining:  1.7min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 16.7min finished
```

Out[39]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators':
[100, 200, 500, 1000, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'sub
sample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

In [40]:

```
print (random_cfl.best_params_)
```

```
{'subsample': 0.5, 'n_estimators': 2000, 'max_depth': 10, 'learning_rate': 0.01,
'colsample_bytree': 0.5}
```

In [42]:

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data
```

```
# find more about XGBClassifier function here
http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGBClassifier
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0,
min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0,
reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbo
se=True, xgb_model=None)
# get_params([deep]) Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is no
t thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-en
sembles/
# -----------------------

x_cfl=XGBClassifier(n_estimators=2000,subsample=0.5,learning_rate=0.01,colsample_bytree=0.5,max_dep
th=10)
x_cfl.fit(X_train_asm,y_train_asm)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_asm,y_train_asm)

predict_y = c_cfl.predict_proba(X_train_asm)
print ('train loss',log_loss(y_train_asm, predict_y))
predict_y = c_cfl.predict_proba(X_cv_asm)
print ('cv loss',log_loss(y_cv_asm, predict_y))
predict_y = c_cfl.predict_proba(X_test_asm)
print ('test loss',log_loss(y_test_asm, predict_y))
```

```
train loss 0.010626478719576738
cv loss 0.033016089856804966
test loss 0.04082419520278345
```

4.5. Machine Learning models on features of both .asm and .bytes files

4.5.1. Merging both asm and byte file features

In [72]:

```
result.head()
```

Out[72]:

	ID	0	1	2	3	4	5	6	7	8	...	6045	24
0	1V2h8lnzeGiuxmHR9k5Q.txt	0.003664	0.005928	0.001729	0.001764	0.001881	0.001747	0.001748	0.002892	0.002843	...	NaN	Na
1	7KHscjvztoka0QpqYFxb.txt	0.001129	0.000376	0.000122	0.000144	0.000169	0.000123	0.000112	0.000156	0.000210	...	NaN	Na
2	2qpZmcvFs4LCBNi9lX6H.txt	0.044132	0.001917	0.000332	0.000474	0.000630	0.000551	0.000275	0.000449	0.000597	...	NaN	Na
3	81jMeyEZiOpJkCuBlDYg.txt	0.001635	0.001525	0.000596	0.000650	0.000848	0.000595	0.000573	0.000936	0.000968	...	NaN	Na
4	5H40FbQlckCMa3jguwS9.txt	0.003595	0.005955	0.001713	0.001739	0.001933	0.001796	0.001627	0.002764	0.002876	...	NaN	Na

5 rows × 516 columns

In [69]:

```
result_asm.head()
```

Out[69]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:	...	esi	eax	ebx	ecx	edi	ebp	es
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3	...	66	15	43	83	0	17	4
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3	...	29	48	82	12	0	14	
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3	...	42	10	67	14	0	11	
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3	...	8	14	7	2	0	8	
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3	...	9	18	29	5	0	11	

5 rows × 54 columns

In [23]:

```
print(result.shape)
print(result_asm.shape)
```

```
(10868, 261)
(10868, 54)
```

In [76]:

```
for index, row in result.iterrows():
    row['ID'].replace('.txt','')
result
```

Out[76]:

	ID	0	1	2	3	4	5	6	7	8	...	60
0	1V2h8lnzeGiuxmHR9k5Q.txt	0.003664	0.005928	0.001729	0.001764	0.001881	0.001747	0.001748	0.002892	0.002843	...	Na
1	7KHscjvztoka0QpqYFxb.txt	0.001129	0.000376	0.000122	0.000144	0.000169	0.000123	0.000112	0.000156	0.000210	...	Na
2	2qpZmcvFs4LCBNi9lX6H.txt	0.044132	0.001917	0.000332	0.000474	0.000630	0.000551	0.000275	0.000449	0.000597	...	Na
3	81jMeyEZiOpJkCuBlDYg.txt	0.001635	0.001525	0.000596	0.000650	0.000848	0.000595	0.000573	0.000936	0.000968	...	Na
4	5H40FbQlckCMa3jguwS9.txt	0.003595	0.005955	0.001713	0.001739	0.001933	0.001796	0.001627	0.002764	0.002876	...	Na
...	...	...	...	...	...	...	...	...	...	...	...	
10862	hZfoeEcsTJRnNLIF5wG0.txt	0.045869	0.019659	0.007080	0.006961	0.008354	0.011321	0.008596	0.011795	0.012075	...	Na
10863	dMr6GYzt8pOD5cQTi2VS.txt	0.004956	0.007812	0.001893	0.001768	0.002043	0.001816	0.001793	0.002990	0.002874	...	Na
10864	eT5zyaVgSRMwGc2oQfHW.txt	0.005040	0.007828	0.001765	0.001827	0.002098	0.001820	0.001845	0.003062	0.002847	...	Na
10865	1YnwQBXNIKPzcdGWxSA6.txt	0.005006	0.007729	0.001776	0.001746	0.001985	0.001868	0.001811	0.002969	0.002763	...	Na
10866	9pVfLJrSuU8iANWPFM15.txt	0.005021	0.007733	0.001810	0.001851	0.002020	0.001828	0.001810	0.002944	0.002971	...	Na

10867 rows × 516 columns

In [21]:

```
result_x = pd.merge(result,result_asm,on='ID', how='left')
result_y = result_x['Class']
result_x = result_x.drop(['ID','rtn','.BSS:','.CODE','Class'], axis=1)
result_x.head()
```

Out[21]:

	Unnamed: 0	0	1	2	3	4	5	6	7	8	...	edx	esi	eax	ebx	
0	0.000000	0.003664	0.005928	0.001729	0.001764	0.001881	0.001747	0.001748	0.002892	0.002843	...	91	135	200	190	
1	0.000092	0.001129	0.000376	0.000122	0.000144	0.000169	0.000123	0.000112	0.000156	0.000210	...	191	198	150	104	
2	0.000184	0.044132	0.001917	0.000332	0.000474	0.000630	0.000551	0.000275	0.000449	0.000597	...	1114	3211	1638	883	1
3	0.000276	0.001635	0.001525	0.000596	0.000650	0.000848	0.000595	0.000573	0.000936	0.000968	...	67	60	53	16	
4	0.000368	0.003595	0.005955	0.001713	0.001739	0.001933	0.001796	0.001627	0.002764	0.002876	...	104	106	216	185	

5 rows × 307 columns

In [22]:

```
result_y.head()
```

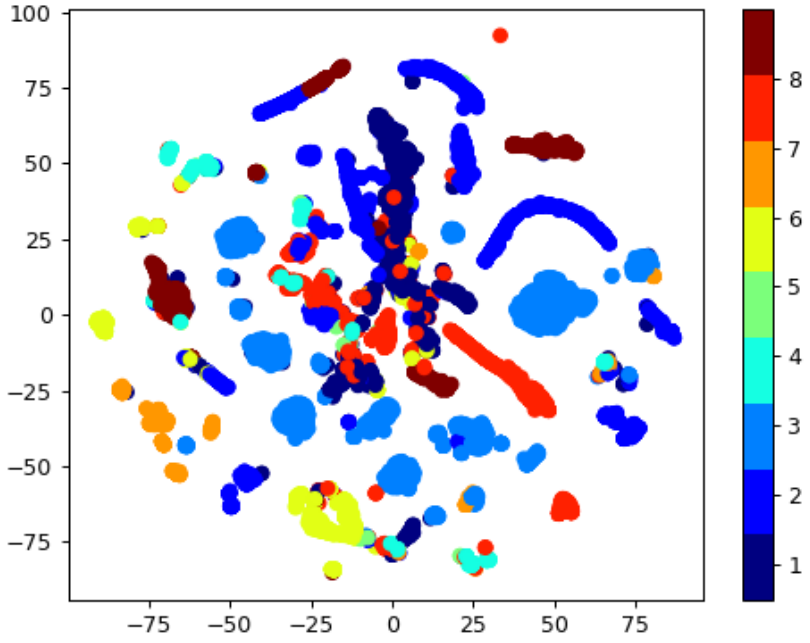
Out[22]:

```
0    3
1    4
2    9
3    4
4    3
Name: Class, dtype: int64
```

4.5.2. Multivariate Analysis on final fearures

In [25]:

```
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result_x)
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=result_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(9))
plt.clim(0.5, 9)
plt.show()
```



4.5.3. Train and Test split

In [26]:

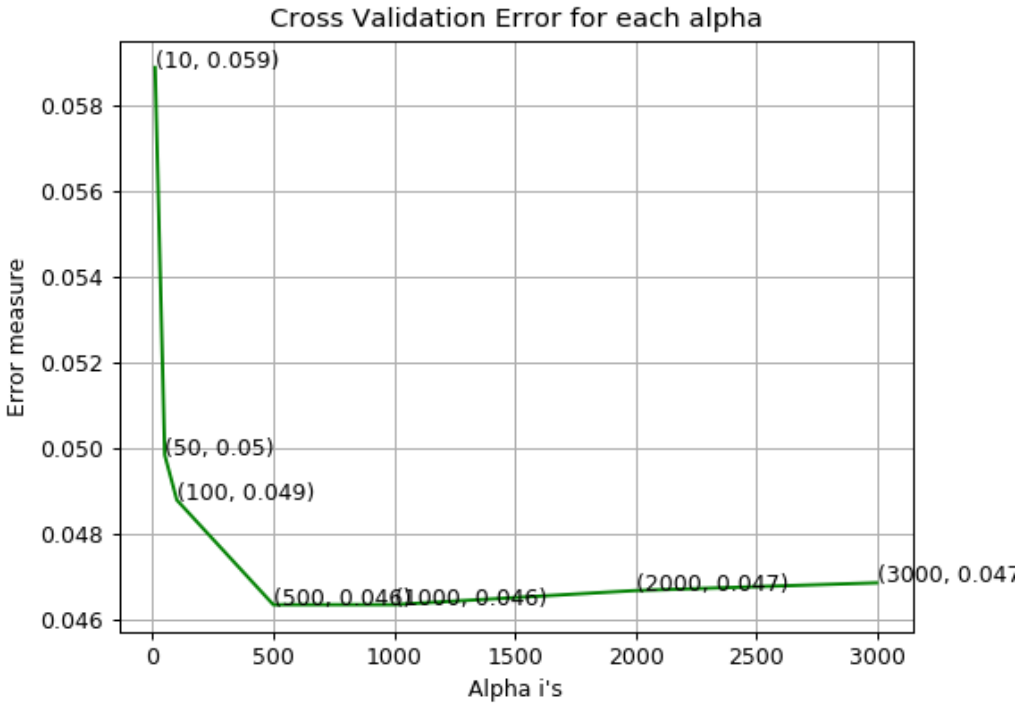
```
X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stratify=result_
y,test_size=0.20)
X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_train,stratify=y
_train,test_size=0.20)
```

4.5.4. Random Forest Classifier on final features

In [34]:

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_s
amples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_
impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None,
verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# predict_proba (X) Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-fores
t-and-their-construction-2/
# --------------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train_merge,y_train_merge)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_merge, y_train_merge)
    predict_y = sig_clf.predict_proba(X_cv_merge)
    cv_log_error_array.append(log_loss(y_cv_merge, predict_y, labels=r_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train_merge,y_train_merge)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss
is:",log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv_merge, predict_y))
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss
is:",log_loss(y_test_merge, predict_y))
```

```
log_loss for c =  10 is 0.058864988008900165
log_loss for c =  50 is 0.04982171352389583
log_loss for c =  100 is 0.04877439563993806
log_loss for c =  500 is 0.04633136949419593
log_loss for c =  1000 is 0.04633282669842955
log_loss for c =  2000 is 0.04666148931304081
log_loss for c =  3000 is 0.04684161733430787
```




```
For values of best alpha =  500 The train log loss is: 0.015045746557915482
For values of best alpha =  500 The cross validation log loss is: 0.04633136949419593
For values of best alpha =  500 The test log loss is: 0.0419437056294099
```

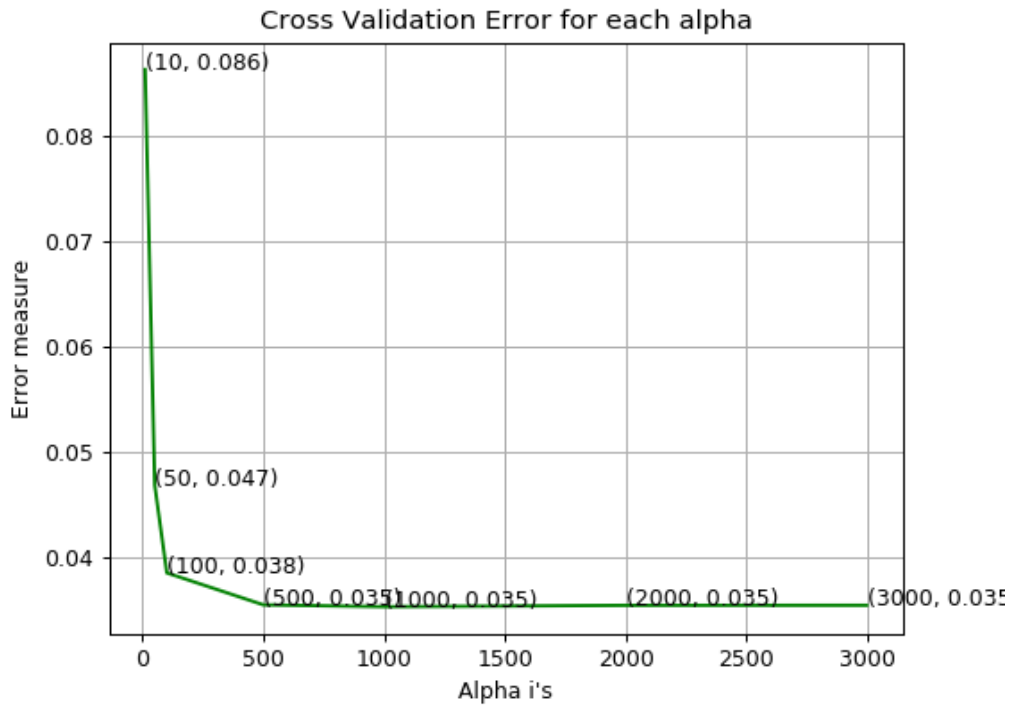
4.5.5. XgBoost Classifier on final features

In [35]:

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here
http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGBClassifier
# ------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0,
min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0,
reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbo
se=True, xgb_model=None)
# get_params([deep]) Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is no
t thread safe.
# get_score(importance_type='weight') -> get the feature importance
# ------------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-en
sembles/
# ------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i)
    x_cfl.fit(X_train_merge,y_train_merge)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_merge, y_train_merge)
    predict_y = sig_clf.predict_proba(X_cv_merge)
    cv_log_error_array.append(log_loss(y_cv_merge, predict_y, labels=x_cfl.classes_, eps=1e-15))
```

```
for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

x_cfl=XGBClassifier(n_estimators=3000,nthread=-1)
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss
is:",log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv_merge, predict_y))
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss
is:",log_loss(y_test_merge, predict_y))
```

```
log_loss for c =  10 is 0.08634410259197668
log_loss for c =  50 is 0.0467962200270487
log_loss for c =  100 is 0.03846464669244138
log_loss for c =  500 is 0.03542509345482663
log_loss for c =  1000 is 0.03524790113745623
log_loss for c =  2000 is 0.03537820448736872
log_loss for c =  3000 is 0.035384159245550155
```




```
For values of best alpha =  1000 The train log loss is: 0.010771162453744454
For values of best alpha =  1000 The cross validation log loss is: 0.035384159245550155
For values of best alpha =  1000 The test log loss is: 0.024834218493213808
```

4.5.5. XgBoost Classifier on final features with best hyper parameters using Random search

In [36]:

```
x_cfl=XGBClassifier()

prams={
    'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
     'n_estimators':[100,200,500,1000,2000],
     'max_depth':[3,5,10],
    'colsample_bytree':[0.1,0.3,0.5,1],
    'subsample':[0.1,0.3,0.5,1]
}
random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
random_cfl.fit(X_train_merge, y_train_merge)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 tasks      | elapsed:  4.6min
[Parallel(n_jobs=-1)]: Done  10 tasks      | elapsed: 11.0min
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed: 17.5min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 28.6min remaining:  3.2min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 37.0min finished
```

Out[36]:

```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
          estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=True, subsample=1),
          fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
          param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_estimators':
[100, 200, 500, 1000, 2000], 'max_depth': [3, 5, 10], 'colsample_bytree': [0.1, 0.3, 0.5, 1], 'sub
sample': [0.1, 0.3, 0.5, 1]},
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          return_train_score='warn', scoring=None, verbose=10)
```

In [37]:

```
print (random_cfl.best_params_)
```

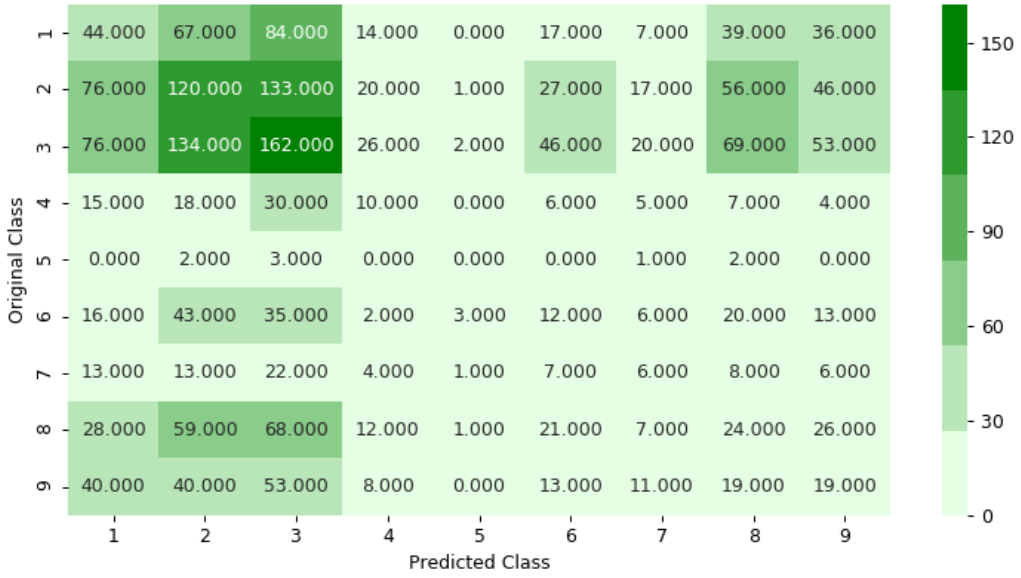
```
{'subsample': 1, 'n_estimators': 200, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 0.
5}
```

In [39]:

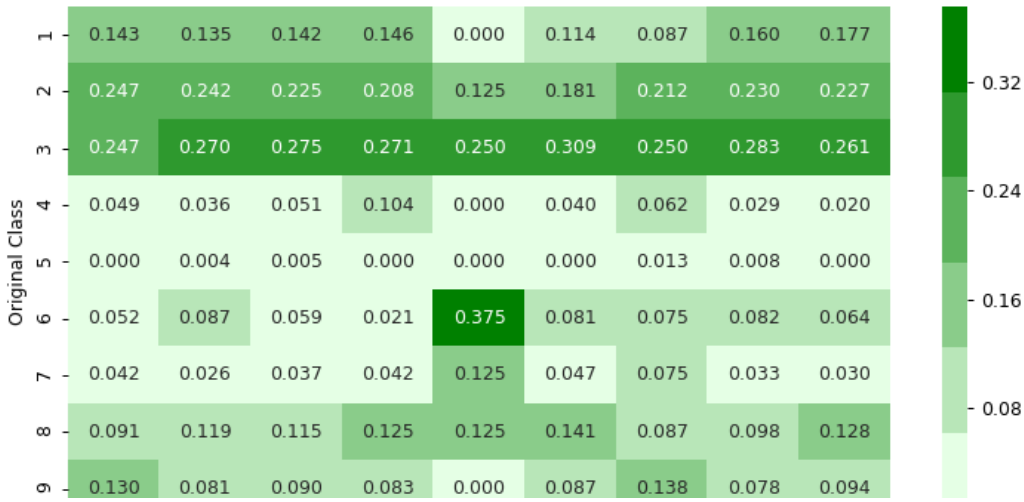
```
# find more about XGBClassifier function here
http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGBClassifier
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0,
min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0,
reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbo
se=True, xgb_model=None)
# get_params([deep]) Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is no
t thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -------------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-en
sembles/
# ------------------------
```

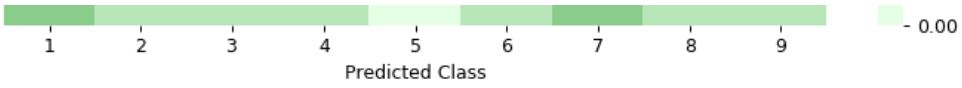
```
x_cfl=XGBClassifier(n_estimators=1000,max_depth=5,learning_rate=0.1,colsample_bytree=0.5,subsample
=1,nthread=-1)
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss
is:",log_loss(y_train_merge, predict_y))
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv_merge, predict_y))
predict_y = sig_clf.predict_proba(X_test_merge)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss
is:",log_loss(y_test_merge, predict_y))
plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_merge))
```

```
For values of best alpha =  1000 The train log loss is: 0.010849938040281054
For values of best alpha =  1000 The cross validation log loss is: 0.03269108838914283
For values of best alpha =  1000 The test log loss is: 0.02814277993749233
Number of misclassified points  81.73873045078197
---------------------------------------------------- Confusion matrix ------------------------------
----------------
```

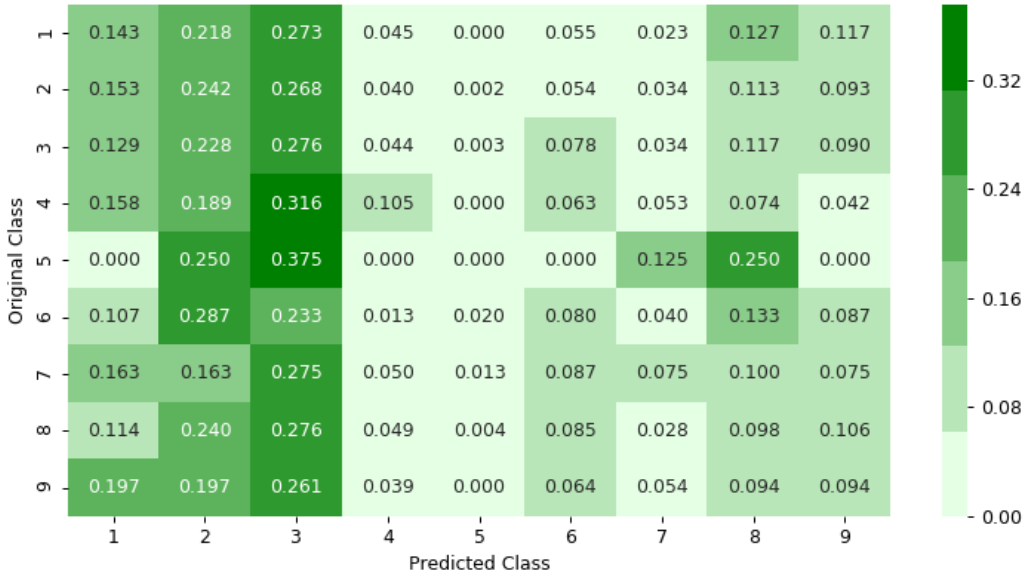



```
---------------------------------------------------- Precision matrix ------------------------------
----------------
```






```
Sum of columns in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
-------------------------------------------------- Recall matrix ----------------------------------
-------------
```




```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

byte features

In [132]:

```
result_x['ID'] = result.ID
```

In [133]:

```
byte_vocab =
"00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,18,19,1a,1b,1c,1d,1e,1f,2(
22,23,24,25,26,27,28,29,2a,2b,2c,2d,2e,2f,30,31,32,33,34,35,36,37,38,39,3a,3b,3c,3d,3e,3f,40,41,42,
4,45,46,47,48,49,4a,4b,4c,4d,4e,4f,50,51,52,53,54,55,56,57,58,59,5a,5b,5c,5d,5e,5f,60,61,62,63,64,(
,67,68,69,6a,6b,6c,6d,6e,6f,70,71,72,73,74,75,76,77,78,79,7a,7b,7c,7d,7e,7f,80,81,82,83,84,85,86,8
89,8a,8b,8c,8d,8e,8f,90,91,92,93,94,95,96,97,98,99,9a,9b,9c,9d,9e,9f,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,
b,ac,ad,ae,af,b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,ba,bb,bc,bd,be,bf,c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,ca,cb,(
,ce,cf,d0,d1,d2,d3,d4,d5,d6,d7,d8,d9,da,db,dc,dd,de,df,e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,ea,eb,ec,ed,ee
f0,f1,f2,f3,f4,f5,f6,f7,f8,f9,fa,fb,fc,fd,fe,ff,??"
```

In [137]:

```
def byte_bigram():
    byte_bigram_vocab = []
    for i, v in enumerate(byte_vocab.split(',')):
        for j in range(0, len(byte_vocab.split(','))):
            byte_bigram_vocab.append(v + ' ' +byte_vocab.split(',')[j])
    return byte_bigram_vocab
```

In [138]:

```
byte_bigram_vocab = byte_bigram()
```

In [139]:

```
byte_bigram_vocab[:5]
```

Out[139]:

```
['00 00', '00 01', '00 02', '00 03', '00 04']
```

In [30]:

```
def byte_trigram():
    byte_trigram_vocab = []
    for i, v in enumerate(byte_vocab.split(',')):
        for j in range(0, len(byte_vocab.split(','))):
            for k in range(0, len(byte_vocab.split(','))):
                byte_trigram_vocab.append(v + ' ' +byte_vocab.split(',')[j]+' '+byte_vocab.split(',
)[k])
    len(byte_trigram_vocab)
```

In [6]:

```
byte_trigram()
```

Out[6]:

```
16974593
```

In [7]:

```
byte_trigram_vocab[:5]
```

Out[7]:

```
['00 00 00', '00 00 01', '00 00 02', '00 00 03', '00 00 04']
```

In [28]:

```
from tqdm import tqdm
from sklearn.feature_extraction.text import CountVectorizer
```

In [38]:

```
vector = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab)
bytebigram_vect = scipy.sparse.csr_matrix((10868, 66049))
for i, file in tqdm(enumerate(os.listdir('byteFiles'))):
    f = open('byteFiles/' + file)
    a[i:]+= scipy.sparse.csr_matrix(vect.fit_transform([f.read().replace('\n', ' ').lower()]))
    f.close()
```

```
10868it [3:49:23,  2.10it/s]
```

In [39]:

```
bytebigram_vect
```

Out[39]:

```
<10868x66049 sparse matrix of type '<class 'numpy.float64'>'
 with 0 stored elements in Compressed Sparse Row format>
```

In [40]:

```
scipy.sparse.save_npz('bytebigram.npz', bytebigram_vect)
```

In [128]:

```
from sklearn.preprocessing import normalize
byte_bigram_vect = normalize(scipy.sparse.load_npz('./Microsoft Malware Detection/bytebigram.npz')
```

```
, axis = 0)
```

N-Gram(2-Gram, 3-Gram, 4-Gram) Opcode Vectorization

In [33]:

```
opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'dec', 'add','i
mul', 'xchg', 'or', 'shr', 'cmp', 'call', 'shl', 'ror', 'rol', 'jnb','jz','rtn','lea','movzx']
```

In [34]:

```
def asmopcodebigram():
    asmopcodebigram = []
    for i, v in enumerate(opcodes):
        for j in range(0, len(opcodes)):
            asmopcodebigram.append(v + ' ' + opcodes[j])
    return asmopcodebigram
```

In [35]:

```
asmopcodebigram = asmopcodebigram()
```

In [36]:

```
def asmopcodetrigram():
    asmopcodetrigram = []
    for i, v in enumerate(opcodes):
        for j in range(0, len(opcodes)):
            for k in range(0, len(opcodes)):
                asmopcodetrigram.append(v + ' ' + opcodes[j] + ' ' + opcodes[k])
    return asmopcodetrigram
```

In [37]:

```
asmopcodetrigram = asmopcodetrigram()
```

In [38]:

```
def asmopcodetetragram():
    asmopcodetetragram = []
    for i, v in enumerate(opcodes):
        for j in range(0, len(opcodes)):
            for k in range(0, len(opcodes)):
                for l in range(0, len(opcodes)):
                    asmopcodetetragram.append(v + ' ' + opcodes[j] + ' ' + opcodes[k] + ' ' +
opcodes[l])
    return asmopcodetetragram
```

In [39]:

```
asmopcodetetragram = asmopcodetetragram()
```

In [ ]:

```
def opcode_collect():
    op_file = open("opcode_file.txt", "w+")
    for asmfile in os.listdir('asmFiles'):
        opcode_str = ""
        with codecs.open('asmFiles/' + asmfile, encoding='cp1252', errors ='replace') as fli:
            for lines in fli:
                line = lines.rstrip().split()
                for li in line:
                    if li in opcodes:
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()
opcode_collect()
```

In [47]:

```
vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asmopcodebigram)
opcodebivect = scipy.sparse.csr_matrix((10868, len(asmopcodebigram)))
raw_opcode = open('opcode_file.txt').read().split('\n')

for indx in range(10868):
    opcodebivect[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
```

In [48]:

```
opcodebivect
```

Out[48]:

```
<10868x676 sparse matrix of type '<class 'numpy.float64'>'
 with 1877309 stored elements in Compressed Sparse Row format>
```

In [49]:

```
scipy.sparse.save_npz('opcodebigram.npz', opcodebivect)
```

In [51]:

```
vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asmopcodetrigram)
opcodetrivect = scipy.sparse.csr_matrix((10868, len(asmopcodetrigram)))

for indx in range(10868):
    opcodetrivect[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
```

In [52]:

```
opcodetrivect
```

Out[52]:

```
<10868x17576 sparse matrix of type '<class 'numpy.float64'>'
 with 7332672 stored elements in Compressed Sparse Row format>
```

In [53]:

```
scipy.sparse.save_npz('opcodetrigram.npz', opcodetrivect)
```

In [54]:

```
vect = CountVectorizer(ngram_range=(4, 4), vocabulary = asmopcodetetragram)
opcodetetravect = scipy.sparse.csr_matrix((10868, len(asmopcodetetragram)))

for indx in range(10868):
    opcodetetravect[indx, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[indx]]))
```

In [55]:

```
opcodetetravect
```

Out[55]:

```
<10868x456976 sparse matrix of type '<class 'numpy.float64'>'
 with 16605229 stored elements in Compressed Sparse Row format>
```

In [56]:

```
scipy.sparse.save_npz('opcodetetragram.npz', opcodetetravect)
```

In [3]:

```
import scipy
opcodetetravect = scipy.sparse.load_npz('./Microsoft Malware Detection/opcodetetragram.npz')
```

In [4]:

```
opcodetrivect=scipy.sparse.load_npz('./Microsoft Malware Detection/opcodetrigram.npz')
```

In [5]:

```
opcodebivect=scipy.sparse.load_npz('./Microsoft Malware Detection/opcodebigram.npz')
```

Image Feature Extraction From ASM Files

In [35]:

```
import array
```

In [64]:

```
def collect_img_asm():
    for asmfile in os.listdir("asmFiles"):
        filename = asmfile.split('.')[0]
        file = codecs.open("asmFiles/" + asmfile, 'rb')
        filelen = os.path.getsize("asmFiles/" + asmfile)
        width = int(filelen ** 0.5)
        rem = int(filelen / width)
        arr = array.array('B')
        arr.frombytes(file.read())
        file.close()
        reshaped = np.reshape(arr[:width * width], (width, width))
        reshaped = np.uint8(reshaped)
        scipy.misc.imsave('asm_image/' + filename + '.png',reshaped)
```

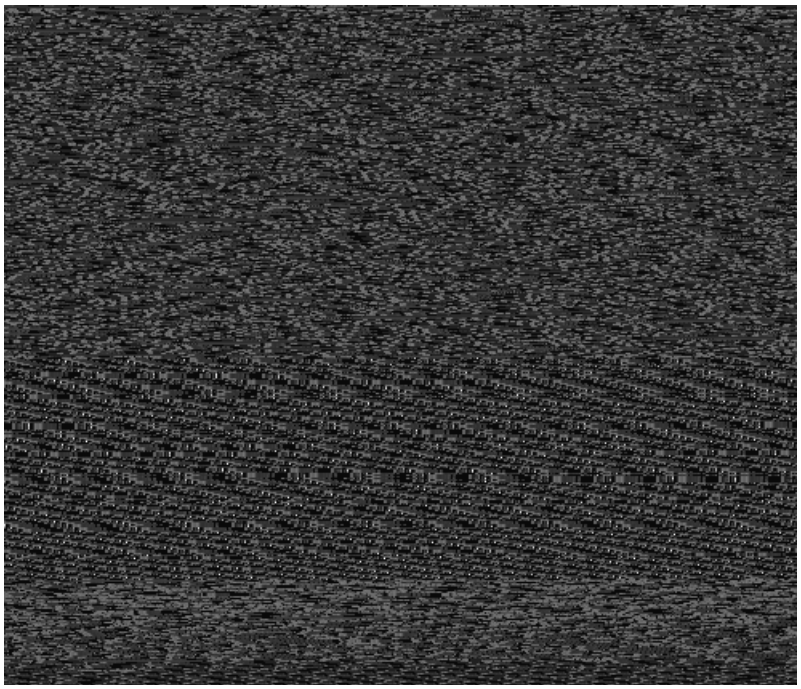
In [65]:

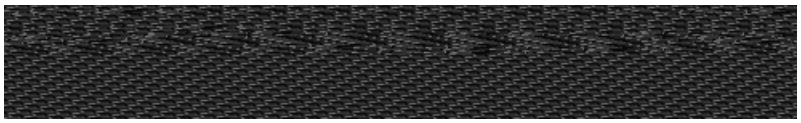
```
collect_img_asm()
```

In [7]:

```
from IPython.display import Image
Image(filename='./Microsoft Malware Detection/asm_imagedeTXH9Zau7qmM0yfYsRS.png')
```

Out[7]:





First 200 Image Pixels

In [8]:

```
import cv2
imagefeatures = np.zeros((10868, 200))
```

In [10]:

```
for i, asmfile in enumerate(os.listdir("./Microsoft Malware Detection/asmFiles")):
    img = cv2.imread("./Microsoft Malware Detection/asm_image" + asmfile.split('.')[0] + '.png')
    img_arr = img.flatten()[:200]
    imagefeatures[i, :] += img_arr
```

In [16]:

```
from sklearn.preprocessing import normalize
imgfeatures_name = []
for i in range(200):
    imgfeatures_name.append('pix' + str(i))
imgdf = pd.DataFrame(normalize(imagefeatures), columns = imgfeatures_name)
```

In [24]:

```
imgdf['ID'] = result.ID
```

In [25]:

```
imgdf.head()
```

Out[25]:

	pix0	pix1	pix2	pix3	pix4	pix5	pix6	pix7	pix8	pix9	...	pix191	pix192	pix19
0	0.107571	0.107571	0.107571	0.103089	0.103089	0.103089	0.097113	0.097113	0.097113	0.101595	...	0.067232	0.067232	0.06723
1	0.107353	0.107353	0.107353	0.102880	0.102880	0.102880	0.096916	0.096916	0.096916	0.101389	...	0.067096	0.067096	0.06709
2	0.107353	0.107353	0.107353	0.102880	0.102880	0.102880	0.096916	0.096916	0.096916	0.101389	...	0.067096	0.067096	0.06709
3	0.107353	0.107353	0.107353	0.102880	0.102880	0.102880	0.096916	0.096916	0.096916	0.101389	...	0.067096	0.067096	0.06709
4	0.107571	0.107571	0.107571	0.103089	0.103089	0.103089	0.097113	0.097113	0.097113	0.101595	...	0.067232	0.067232	0.06723

5 rows × 201 columns

In [27]:

```
import joblib

joblib.dump(imgdf, './Microsoft Malware Detection/img_df')
```

Out[27]:

```
['./Microsoft Malware Detection/img_df']
```

In [38]:

```
import joblib
img_df=joblib.load('./Microsoft Malware Detection/img_df')
```

In [29]:

```
img_df.head()
```

Out[29]:

	pix0	pix1	pix2	pix3	pix4	pix5	pix6	pix7	pix8	pix9	...	pix191	pix192	pix19
0	0.107571	0.107571	0.107571	0.103089	0.103089	0.103089	0.097113	0.097113	0.097113	0.101595	...	0.067232	0.067232	0.06723
1	0.107353	0.107353	0.107353	0.102880	0.102880	0.102880	0.096916	0.096916	0.096916	0.101389	...	0.067096	0.067096	0.06709
2	0.107353	0.107353	0.107353	0.102880	0.102880	0.102880	0.096916	0.096916	0.096916	0.101389	...	0.067096	0.067096	0.06709
3	0.107353	0.107353	0.107353	0.102880	0.102880	0.102880	0.096916	0.096916	0.096916	0.101389	...	0.067096	0.067096	0.06709
4	0.107571	0.107571	0.107571	0.103089	0.103089	0.103089	0.097113	0.097113	0.097113	0.101595	...	0.067232	0.067232	0.06723

5 rows × 201 columns

Important Feature Selection Using Random Forest

In [30]:

```
def imp_features(data, features, keep):
    rf = RandomForestClassifier(n_estimators = 100, n_jobs = -1)
    rf.fit(data, result_y)
    imp_feature_indx = np.argsort(rf.feature_importances_)[::-1]
    imp_value = np.take(rf.feature_importances_, imp_feature_indx[:20])
    imp_feature_name = np.take(features, imp_feature_indx[:20])
    sns.set()
    plt.figure(figsize = (10, 5))
    ax = sns.barplot(x = imp_feature_name, y = imp_value)
    ax.set_xticklabels(labels = imp_feature_name, rotation = 45)
    sns.set_palette(reversed(sns.color_palette("husl", 10)), 10)
    plt.title('Important Features')
    plt.xlabel('Feature Names')
    plt.ylabel('Importance')
    return imp_feature_indx[:keep]
```

Important Feature Among Opcode Bi-Gram

In [106]:

```
normalize(opcodebivect, axis = 0).shape
```

Out[106]:

```
(10868, 676)
```

In [107]:

```
from sklearn.preprocessing import normalize
# asmopcodebigram
# df = pd.DataFrame(normalize(opcodebivect, axis = 0).toarray())
# df.fillna(df.mean(),inplace=True)
# # np.nan_to_num(normalize(opcodebivect, axis = 0))
op_bi_indxes = imp_features(normalize(opcodebivect, axis = 0)[:10867], asmopcodebigram, 200)
```

In [108]:

```
op_bi_df = pd.SparseDataFrame(normalize(opcodebivect, axis = 0), columns = asmopcodebigram)
for col in op_bi_df.columns:
    if col not in np.take(asmopcodebigram, op_bi_indxes):
        op_bi_df.drop(col, axis = 1, inplace = True)
```

In [109]:

```
op_bi_df.to_dense().to_csv('./Microsoft Malware Detection/op_bi.csv')
```

In [32]:

```
op_bi_df = pd.read_csv('./Microsoft Malware Detection/op_bi.csv').drop('Unnamed: 0', axis = 1).fill
na(0)
```

In [111]:

```
op_bi_df['ID'] = result.ID
op_bi_df.head()
```

Out[111]:

	jmp jmp	jmp mov	jmp push	jmp pop	jmp xor	jmp sub	jmp add	jmp cmp	jmp call	jmp lea	...	lea add	lea or	lea cmp	
0	0.000578	0.001927	0.003012	0.000749	0.003073	0.0	0.002956	0.005164	0.003385	0.000652	...	0.002505	0.001138	0.004301	0.
1	0.000000	0.000020	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	...	0.000835	0.002845	0.001195	0.
2	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	...	0.000278	0.002276	0.000956	0.
3	0.000000	0.000000	0.000070	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	...	0.001391	0.001138	0.001434	0.
4	0.000000	0.000000	0.000070	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.000000	...	0.000278	0.000000	0.000000	0.

5 rows × 201 columns

Important Feature Among Opcode 3-Gram

In [114]:

```
op_tri_indxes = imp_features(normalize(opcodetrivect, axis = 0)[:10867], asmopcodetrigram, 200)
```

In [115]:

```
op_tri_df = pd.SparseDataFrame(normalize(opcodetrivect, axis = 0), columns = asmopcodetrigram)
op_tri_df = op_tri_df.loc[:, np.intersect1d(op_tri_df.columns, np.take(asmopcodetrigram,
op_tri_indxes))]
```

In [116]:

```
op_tri_df.to_dense().to_csv('./Microsoft Malware Detection/op_tri.csv')
```

In [33]:

```
op_tri_df = pd.read_csv('./Microsoft Malware Detection/op_tri.csv').drop('Unnamed: 0', axis =
1).fillna(0)
```

In [118]:

```
op_tri_df['ID'] = result.ID
op_tri_df.head()
```

Out[118]:

	add add add	add add cmp	add add jmp	add add pop	add add sub	add cmp mov	add cmp push	add jmp push	add lea push	add mov mov	...	xchg mov pop	xor cmp jz	xor lea push	xor r r
0	0.0001	0.004671	0.010089	0.000000	0.000000	0.001945	0.0049	0.003510	0.000397	0.000525	...	0.0	0.003654	0.004995	0.000
1	0.0000	0.000000	0.002522	0.000000	0.000000	0.000000	0.0000	0.000000	0.000000	0.000040	...	0.0	0.000000	0.000000	0.000
2	0.0000	0.000000	0.000000	0.000000	0.000662	0.000000	0.0000	0.000000	0.000000	0.000040	...	0.0	0.000000	0.000000	0.000
3	0.0000	0.000000	0.002522	0.000000	0.000000	0.000000	0.0000	0.000878	0.000397	0.000000	...	0.0	0.000000	0.001665	0.000
4	0.0000	0.000000	0.000000	0.001891	0.002647	0.000000	0.0000	0.000878	0.000000	0.000000	...	0.0	0.000000	0.000000	0.000

5 rows × 201 columns

Important Feature Among Opcode 4-Gram

In [119]:

```
op_tetra_indxes = imp_features(normalize(opcodetetravect, axis = 0)[:10867], asmopcodetetragram,
200)
```

In [120]:

```
op_tetra_df = pd.SparseDataFrame(normalize(opcodetetravect, axis = 0), columns = asmopcodetetragram
)
op_tetra_df = op_tetra_df.loc[:, np.intersect1d(op_tetra_df.columns, np.take(asmopcodetetragram, op
_tetra_indxes))]
```

In [121]:

```
op_tetra_df.to_dense().to_csv('./Microsoft Malware Detection/op_tetra.csv')
```

In [34]:

```
op_tetra_df = pd.read_csv('./Microsoft Malware Detection/op_tetra.csv').drop('Unnamed: 0', axis = 1
).fillna(0)
```

In [123]:

```
op_tetra_df['ID'] = result.ID
op_tetra_df.head()
```

Out[123]:

	add mov push call	add pop call add	add pop lea push	add pop mov push	add pop push call	add pop push lea	add pop push mov	add pop push push	add pop retn mov	add pop retn pop	...	sub mov sub push	sub mov xor push	sub push mov push	p p
0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000785	0.006
1	0.0	0.0	0.022985	0.030729	0.019308	0.015570	0.016022	0.009613	0.023052	0.000000	...	0.007263	0.008773	0.000785	0.000
2	0.0	0.0	0.000000	0.028365	0.016337	0.015570	0.020392	0.009613	0.023052	0.001681	...	0.014527	0.002193	0.000785	0.000
3	0.0	0.0	0.011493	0.023638	0.000000	0.023355	0.027675	0.005341	0.018356	0.000000	...	0.000000	0.004387	0.000000	0.000
4	0.0	0.0	0.000000	0.004728	0.000000	0.000000	0.000000	0.002136	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000

5 rows × 201 columns

Important Feature Among Byte Bi-Gram

In [140]:

```
byte_bi_indxes = imp_features(normalize(byte_bigram_vect, axis = 0)[:10867], byte_bigram_vocab, 300
)
```

In [141]:

```
np.save('./Microsoft Malware Detection/byte_bi_indx', byte_bi_indxes)
```

In [2]:

```
byte_bi_indxes = np.load('./Microsoft Malware Detection/byte_bi_indx.npy')
```

In [35]:

```
top_byte_bi = np.zeros((10868, 0))
for i in byte_bi_indxes:
```

```
    sliced = byte_bigram_vect[:, i].todense()
    top_byte_bi = np.hstack([top_byte_bi, sliced])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-35-59114781a1ab> in <module>
      1 top_byte_bi = np.zeros((10868, 0))
      2 for i in byte_bi_indxes:
----> 3     sliced = byte_bigram_vect[:, i].todense()
      4     top_byte_bi = np.hstack([top_byte_bi, sliced])

NameError: name 'byte_bigram_vect' is not defined
```

In [145]:

```
byte_bi_df = pd.SparseDataFrame(top_byte_bi, columns = np.take(byte_bigram_vocab, byte_bi_indxes))
```

In [146]:

```
byte_bi_df.to_dense().to_csv('./Microsoft Malware Detection/byte_bi.csv')
```

In [4]:

```
byte_bi_df = pd.read_csv('./Microsoft Malware Detection/byte_bi.csv').drop('Unnamed: 0', axis = 1).
fillna(0)
```

In [22]:

```
byte_bi_df['ID'] = result.ID
```

In [23]:

```
byte_bi_df.head()
```

Out[23]:

	?? ??	55 95	55 b3	55 b2	55 b1	55 b0	55 af	55 ae	55 ad	55 ac	...	54 b3	54 b4	54 c4	54 d1	54 d0	54 cf	54 ce	54 cd	54 cc	ID
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1V2h8lnzeGiuxmHR9k5Q
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	7KHscjvztoka0QpqYFxb
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2qpZmcvFs4LCBNi9lX6H
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	81jMeyEZiOpJkCuBlDYg
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5H40FbQlckCMa3jguwS9

5 rows × 301 columns

Advanced features

Adding 300 bytebigram,200 opcode bigram,200 opcode trigram,200 opcode tetragram ,first 200 image pixels

In [39]:

```
final_data = pd.concat([result_x, op_bi_df, op_tri_df, op_tetra_df, byte_bi_df,img_df], axis = 1, j
oin = 'inner')
```

In [151]:

```
final_data = final_data.drop('ID', axis = 1)
```

In [152]:

```
final_data.head()
```

Out[152]:

	0	1	2	3	4	5	6	7	8	9	...	pix190	pix191	pix192	pix193	pix194	pix195	
0	8391	4211	3108	3270	3072	3087	3100	3143	3195	3108	...	0.067232	0.067232	0.067232	0.067232	0.067232	0.067232	0.
1	2585	267	220	267	276	217	199	170	236	211	...	0.067096	0.067096	0.067096	0.067096	0.067096	0.067096	0.
2	101076	1362	597	879	1029	973	487	488	671	487	...	0.067096	0.067096	0.067096	0.067096	0.067096	0.067096	0.
3	3745	1083	1071	1205	1386	1052	1017	1017	1088	1032	...	0.067096	0.067096	0.067096	0.067096	0.067096	0.067096	0.
4	8234	4230	3079	3223	3158	3173	2886	3003	3232	3089	...	0.067232	0.067232	0.067232	0.067232	0.067232	0.067232	0.

5 rows × 1408 columns

In [153]:

```
final_data.to_csv('./Microsoft Malware Detection/final_data.csv')
```

In [2]:

```
final_data = pd.read_csv('./Microsoft Malware Detection/final_data.csv')
```

In [3]:

```
final_data
```

Out[3]:

	Unnamed: 0	0	1	2	3	4	5	6	7	8	...	pix190	pix191	pix192	pix193	pix
0	0	8391	4211	3108	3270	3072	3087	3100	3143	3195	...	0.067232	0.067232	0.067232	0.067232	0.067
1	1	2585	267	220	267	276	217	199	170	236	...	0.067096	0.067096	0.067096	0.067096	0.067
2	2	101076	1362	597	879	1029	973	487	488	671	...	0.067096	0.067096	0.067096	0.067096	0.067
3	3	3745	1083	1071	1205	1386	1052	1017	1017	1088	...	0.067096	0.067096	0.067096	0.067096	0.067
4	4	8234	4230	3079	3223	3158	3173	2886	3003	3232	...	0.067232	0.067232	0.067232	0.067232	0.067
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10862	10862	105055	13964	12725	12904	13646	20003	15248	12817	13570	...	0.066207	0.066207	0.066207	0.066207	0.066
10863	10863	11351	5549	3402	3277	3338	3208	3180	3249	3230	...	0.067096	0.067096	0.067096	0.067096	0.067
10864	10864	11543	5560	3173	3386	3427	3216	3272	3327	3199	...	0.067096	0.067096	0.067096	0.067096	0.067
10865	10865	11465	5490	3192	3236	3242	3300	3212	3226	3105	...	0.067096	0.067096	0.067096	0.067096	0.067
10866	10866	11500	5493	3253	3432	3300	3230	3210	3199	3339	...	0.067096	0.067096	0.067096	0.067096	0.067

10867 rows × 1409 columns

In [23]:

```
x_train_final, x_test_final, y_train_final, y_test_final = train_test_split(final_data, result_y, s
tratify = result_y, test_size = 0.20)
x_trn_final, x_cv_final, y_trn_final, y_cv_final = train_test_split(x_train_final, y_train_final, s
tratify = y_train_final, test_size = 0.20)
```

Machine Learning Models on ASM Features + Byte Features + Advanced Features

In [43]:

```
alpha = [10 ** x for x in range(-5, 4)]
cv_log_error_array=[]
for i in alpha:
```

```
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(x_trn_final,y_trn_final)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(x_trn_final,y_trn_final)
    predict_y = sig_clf.predict_proba(x_cv_final)
    cv_log_error_array.append(log_loss(y_cv_final, predict_y, labels=logisticR.classes_, eps=1e-15)
)

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
log_loss for c =  1e-05 is 0.7906210918232409
log_loss for c =  0.0001 is 0.7998771770734955
log_loss for c =  0.001 is 0.786881493866022
log_loss for c =  0.01 is 0.7879840262964852
log_loss for c =  0.1 is 0.789877468723614
log_loss for c =  1 is 0.7955128056312036
log_loss for c =  10 is 0.7701806228215742
log_loss for c =  100 is 0.7913968691914951
log_loss for c =  1000 is 0.7896696329851713
```

In [44]:

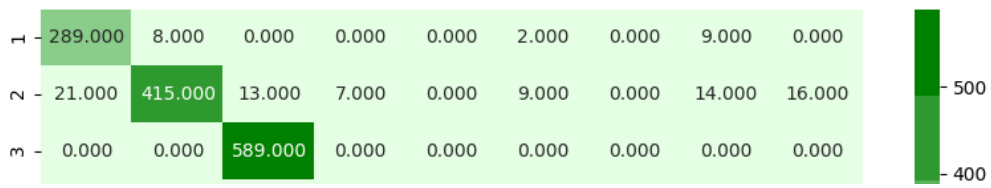
```
logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balanced')
logisticR.fit(x_trn_final,y_trn_final)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(x_trn_final,y_trn_final)

predict_y = sig_clf.predict_proba(x_trn_final)
print ('log loss for train data',(log_loss(y_trn_final, predict_y, labels=logisticR.classes_, eps=1
e-15)))
predict_y = sig_clf.predict_proba(x_cv_final)
print ('log loss for cv data',(log_loss(y_cv_final, predict_y, labels=logisticR.classes_, eps=1e-15
)))
predict_y = sig_clf.predict_proba(x_test_final)
print ('log loss for test data',(log_loss(y_test_final, predict_y, labels=logisticR.classes_, eps=1
e-15)))
```

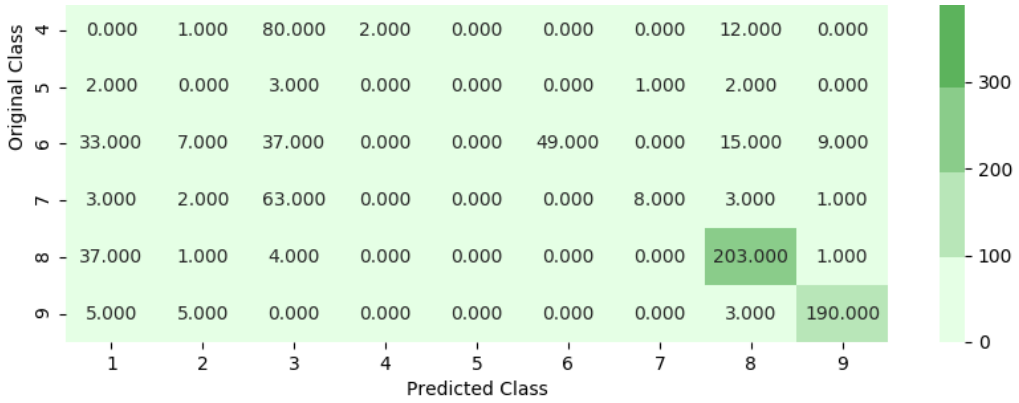
```
log loss for train data 0.6971772763370007
log loss for cv data 0.7701806228215742
log loss for test data 0.7721596566800629
```

In [50]:

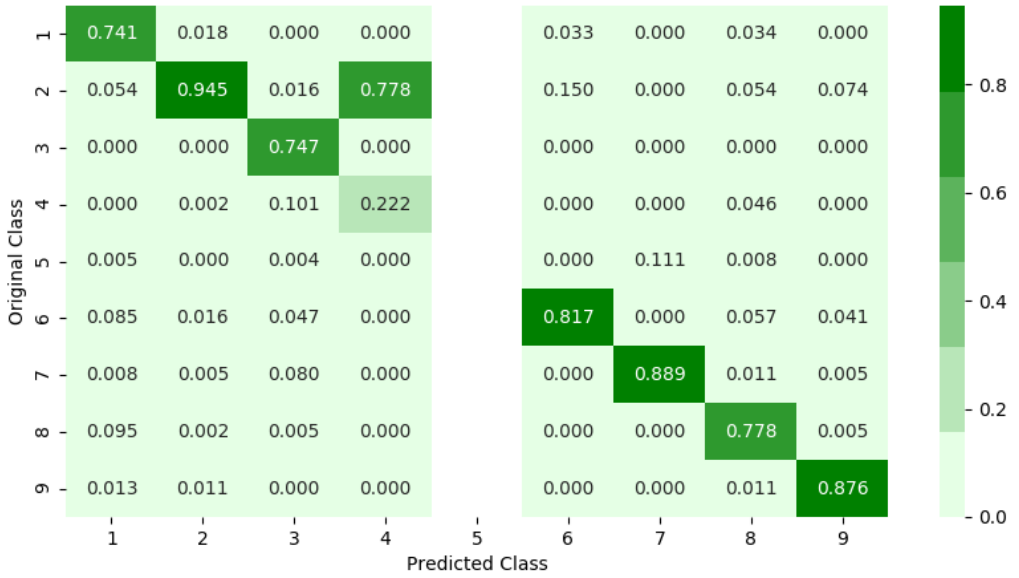
```
plot_confusion_matrix(y_test_final,sig_clf.predict(x_test_final))
```

```
Number of misclassified points  19.733210671573136
---------------------------------------------------- Confusion matrix -----------------------------
----------------
```



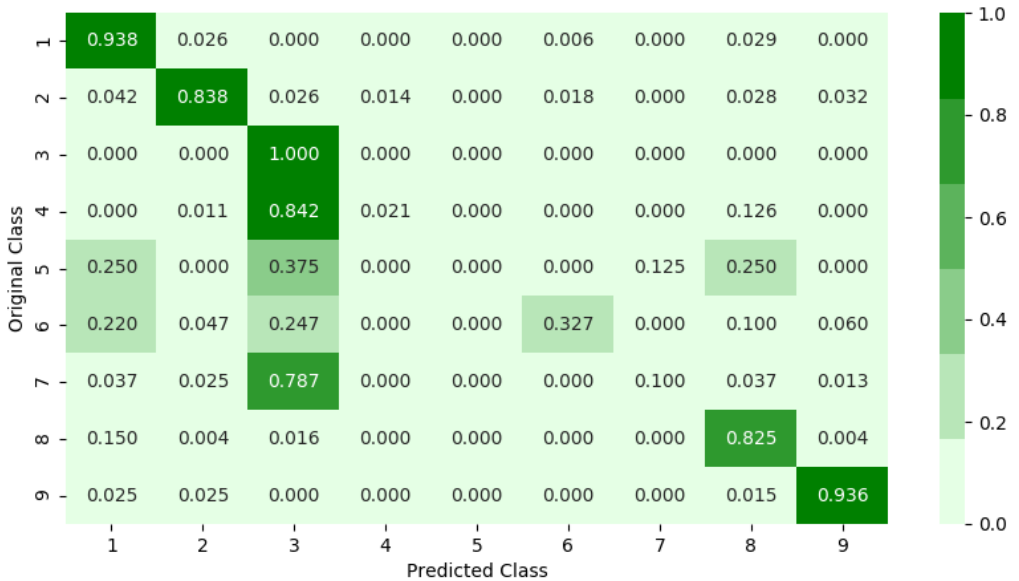


---------- Precision matrix ----------



Sum of columns in precision matrix [1. 1. 1. 1. nan 1. 1. 1. 1.]

---------- Recall matrix ----------



```
Sum of rows in precision matrix [1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

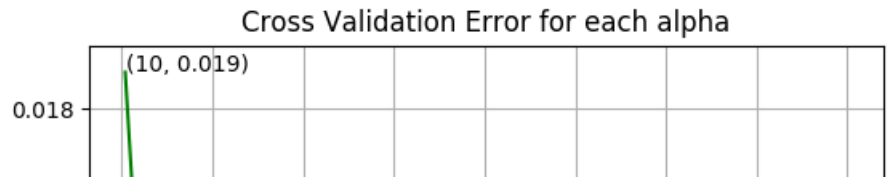
In [ ]:

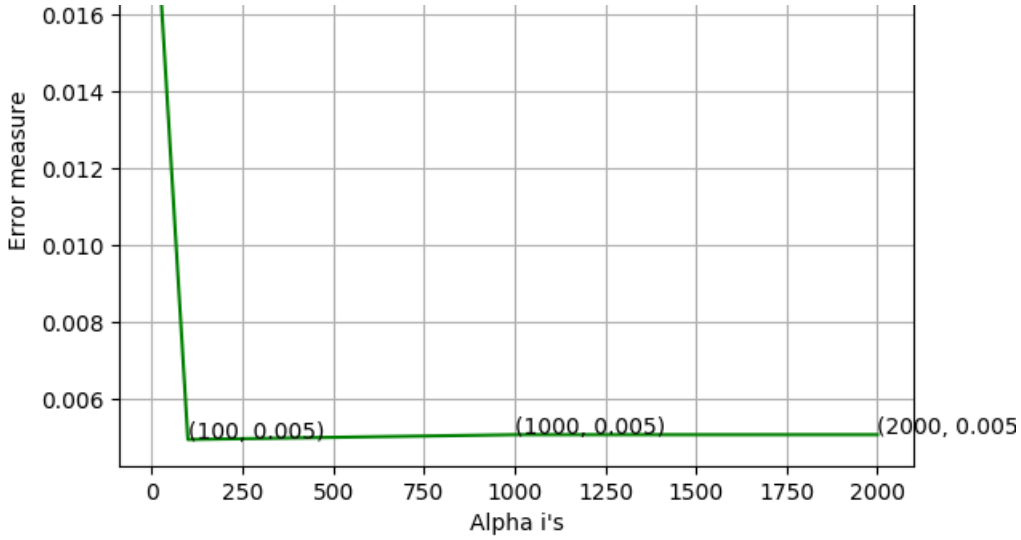
```
```

In [25]:

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

# find more about XGBClassifier function here
http://xgboost.readthedocs.io/en/latest/python/python_api.html?#xgboost.XGBClassifier
# -------------------------
# default paramters
# class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True,
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0,
min_child_weight=1,
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0,
reg_lambda=1,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwargs)

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=None, verbo
se=True, xgb_model=None)
# get_params([deep]) Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This function is no
t thread safe.
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/what-are-en
sembles/
# ------------------------

alpha=[10,100,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i)
    x_cfl.fit(x_trn_final,y_trn_final)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(x_trn_final, y_trn_final)
    predict_y = sig_clf.predict_proba(x_cv_final)
    cv_log_error_array.append(log_loss(y_cv_final, predict_y, labels=x_cfl.classes_, eps=1e-15))

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
log_loss for c =  10 is 0.01895853341098602
log_loss for c =  100 is 0.004947100592248261
log_loss for c =  1000 is 0.005071891309869607
log_loss for c =  2000 is 0.005072350518711517
```





In [26]:

```
x_cfl=XGBClassifier(n_estimators=1000,nthread=-1)
x_cfl.fit(x_trn_final,y_trn_final,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(x_trn_final, y_trn_final)

predict_y = sig_clf.predict_proba(x_trn_final)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss
is:",log_loss(y_trn_final, predict_y))
predict_y = sig_clf.predict_proba(x_cv_final)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_lo
ss(y_cv_final, predict_y))
predict_y = sig_clf.predict_proba(x_test_final)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss
is:",log_loss(y_test_final, predict_y))
```

```
For values of best alpha =  100 The train log loss is: 0.004803184968130076
For values of best alpha =  100 The cross validation log loss is: 0.005071891309869607
For values of best alpha =  100 The test log loss is: 0.004980837664893172
```

Results

In [27]:

```
from prettytable import PrettyTable
ptable = PrettyTable()
ptable.title = " Model Comparision "
ptable.field_names = ["Model",'Features','log loss']
ptable.add_row(["random","Byte files","2.499"])
ptable.add_row(["knn","Byte files","0.485"])
ptable.add_row(["Logistic Regression","Byte files","0.529"])
ptable.add_row(["Random Forest Classifier ","Byte files","0.066"])
ptable.add_row(["XgBoost Classification","Byte files","0.078"])
ptable.add_row(["XgBoost Classification with HyperParameter Tunining","Byte files","0.077"])
ptable.add_row(["\n","\n","\n"])
ptable.add_row(["knn","asmfiles","0.213"])
ptable.add_row(["Logistic Regression","asmfiles","0.383"])
ptable.add_row(["Random Forest Classifier ","asmfiles","0.039"])
ptable.add_row(["XgBoost Classification","asmfiles","0.043"])
ptable.add_row(["XgBoost Classification with HyperParameter Tunining","asmfiles","0.041"])
ptable.add_row(["\n","\n","\n"])
ptable.add_row(["Random Forest Classifier ","Byte files+asmfiles","0.042"])
ptable.add_row(["XgBoost Classification","Byte files+asmfiles","0.025"])
ptable.add_row(["XgBoost Classification with HyperParameter Tunining","Byte files+asmfiles","0.028
"])
ptable.add_row(["\n","\n","\n"])
ptable.add_row(["Logistic Regression","Byte files+asmfiles+advanced features","0.772"])
ptable.add_row(["XgBoost Classification","Byte files+asmfiles+advanced features","0.005"])
print(ptable)
```

```
+-----------------------------------------------------+---------------------------------------+----
--+
|                        Model                        |                Features               | log
ss |
+-----------------------------------------------------+---------------------------------------+----
--+
|                        random                       |              Byte files               |  2.
|
|                         knn                         |              Byte files               |  0.
|
|                  Logistic Regression                |              Byte files               |  0.
9   |
|               Random Forest Classifier              |              Byte files               |  0.
6   |
|                 XgBoost Classification              |              Byte files               |  0.
8   |
| XgBoost Classification with HyperParameter Tunining |              Byte files               |
0.077   |
|                                                     |                                       |
|
|                                                     |                                       |
|
|                        knn                          |               asmfiles                |  0.
|
|                  Logistic Regression                |               asmfiles                |  0.
3   |
|               Random Forest Classifier              |               asmfiles                |  0.
9   |
|                 XgBoost Classification              |               asmfiles                |  0.
3   |
| XgBoost Classification with HyperParameter Tunining |               asmfiles                |
0.041   |
|                                                     |                                       |
|
|                                                     |                                       |
|
|               Random Forest Classifier              |          Byte files+asmfiles          |
0.042   |
|                 XgBoost Classification              |          Byte files+asmfiles          |
0.025   |
| XgBoost Classification with HyperParameter Tunining |          Byte files+asmfiles          |  0.
028   |
|                                                     |                                       |
|
|                                                     |                                       |
|
|                  Logistic Regression                | Byte files+asmfiles+advanced features |  0.
72   |
|                 XgBoost Classification              | Byte files+asmfiles+advanced features |  0.
05   |
+-----------------------------------------------------+---------------------------------------+----
--+
```

In [ ]: